Using the JMS Client for vFabric RabbitMQ

vFabric RabbitMQ 3.1

This document supports the version of each product listed and supports all subsequent versions until the document is replaced by a new edition. To check for more recent editions of this document, see http://www.vmware.com/support/pubs.

vmware

You can find the most up-to-date technical documentation on the VMware Web site at: https://www.vmware.com/support/.

The VMware Web site also provides the latest product updates.

If you have comments about this documentation, submit your feedback to: docfeedback@vmware.com

VMware, Inc., 3401 Hillview Avenue, Palo Alto, CA 94304

www.vmware.com

Table of Contents

1. About Using the JMS Client for vFabric RabbitMQ

Revised September 3, 2013.

Using the JMS Client for vFabric RabbitMQ describes using the JMS client with the VMware© vFabric™ RabbitMQ server. Read this documentation to learn how to set up the JMS client software and configure applications to use it.

Intended Audience

This document is for anyone who wishes to access RabbitMQ messaging with existing or new Java Message Service (JMS) applications.

2. About JMS Client for vFabric RabbitMQ

JMS Client for vFabric RabbitMQ is a client library for vFabric RabbitMQ. vFabric RabbitMQ is not a JMS provider but has features needed to support the JMS Queue and Topic messaging models. JMS Client for RabbitMQ implements the JMS 1.1 specification on top of the RabbitMQ Java client API, thus allowing new and existing JMS applications to connect with RabbitMQ brokers through Advanced Message Queueing Protocol (AMQP).

Subtopics

Components of JMS Client for vFabric RabbitMQ

The JMS Client for vFabric RabbitMQ distribution archive file contains the following components:

- JMS Client for RabbitMQ library and its dependent libraries.

  `rabbitmq-jms-`*`version`*`.jar` is the JMS Client for RabbitMQ. The dependent libraries are in the `dependencies` directory. They are `amqp-client-`*`version`*`.jar`, which is the RabbitMQ Java client library, and `geronimo-jms_1.1_spec-`*`version`*`.jar`, which contains the JMS 1.1 interfaces that JMS Client for RabbitMQ implements.

- RabbitMQ JMS topic selector plugin, `plugin/rjms-topic-selector-`*`version`*`.ez`.

  To support message selectors for JMS topics, the RabbitMQ Topic Selector plugin must be installed on the RabbitMQ server. Message selectors allow a JMS application to filter messages using an expression based on SQL syntax. Message selectors for Queues are not currently supported.

Because the RabbitMQ Java client library is included in the JMS Client for RabbitMQ distribution, you do not have to download and install the vFabric RabbitMQ Java Client.

JMS and AMQP

JMS is the standard messaging service for the Java Extended Edition (JEE) platform. It is available in commercial and open source implementations. Each implementation includes a JMS provider, a JMS client library, and additional, implementation-specific components for administering the messaging system. The JMS provider can be a standalone implementation of the messaging service, or a bridge to a non-JMS messaging system.

The JMS client API is standardized, so JMS applications are portable between vendors' implementations. However, the underlying messaging implementation is unspecified, so there is no interoperability between JMS implementations. Java applications that want to share messaging must all use the same JMS implementation unless bridging technology exists. Furthermore, non-Java applications cannot access JMS without a vendor-specific JMS client library to enable interoperability.

AMQP is a messaging protocol, rather than an API like JMS. Any client that implements the protocol can access any AMQP broker. Protocol-level interoperability allows AMQP clients written in any programming language and running on any operating system to participate in the messaging system with no need to bridge incompatible vendor implementations.

Because JMS Client for RabbitMQ is implemented using the RabbitMQ Java client, it is compliant with both the JMS API and the AMQP protocol.

You can download the JMS 1.1 specification and API documentation from the Oracle Technology Network Web site.

Limitations

Some JMS features are unsupported in this JMS Client for RabbitMQ release:

- The JMS Client for RabbitMQ does not support server sessions.
- XA transaction support interfaces are not implemented.
- Topic selectors are supported with the RabbitMQ JMS topic selector plugin. Queue selectors are not yet implemented.
- QueueBrowser is not implemented.
- SSL and socket options for RabbitMQ connections are not implemented.
- The JMS `NoLocal` subscription feature, which prevents delivery of messages published from a subscriber's own connection, is not supported with RabbitMQ. You can call a method that includes the `NoLocal` argument, but it is ignored.

See Chapter 4, *vFabric RabbitMQ Implementation of JMS API* for a detailed list of supported JMS APIs.

3. Installing and Configuring JMS Client for vFabric RabbitMQ

This chapter describes how to install and configure the JMS Client for vFabric RabbitMQ.

Subtopics

Install the JMS Client Software and the Topic Selector Plug-in

The JMS Client for vFabric RabbitMQ is distributed in a ZIP or compressed TAR file.

Download and Install JMS Client for vFabric RabbitMQ

1. Browse to the vFabric RabbitMQ download Web site at https://www.vmware.com/go/download-rmq.
2. Click the Drivers & Tools tab, expand VMware vFabric RabbitMQ, and click the Go to Download button next to JMS Client 1.0.X for VMware vFabric RabbitMQ.
3. Download the `rabbitmq-jms-package-`*`version`*`-client-and-plugin.tar.gz` or `rabbitmq-jms-package-`*`version`*`-client-and-plugin.zip` file.
4. Extract the archive to a directory on your computer, using the `unzip` or `tar` command, or another suitable archive utility.

For information about using the JMS Client for RabbitMQ libraries with an application server or in a Web application, see Configure JMS Applications to Use JMS Client for vFabric RabbitMQ.

Install the vFabric RabbitMQ Topic Selector Plug-in

You install the plug-in in your RabbitMQ server `plugins` directory and enable it. The location of the `plugins` directory depends on your platform and how you installed RabbitMQ.

1. Copy the file `plugin/rjms-topic-selector-`*`version`*`.ez` from the RabbitMQ for JMS Client distribution into the `plugins` directory of your RabbitMQ server installation. For example, if you installed RabbitMQ from an RPM, enter this command in the directory where you extracted the distribution:

   ```
   $ cp plugin/rjms-topic-selector-1.0.0.ez /opt/vmware/rabbitmq/lib/rabbitmq_server-1.0/plugins
   ```

2. Enable the plug-in using the **rabbitmq-plugins** command:

   ```
   $ rabbitmq-plugins enable rjms-topic-selector
   ```

3. Restart the RabbitMQ server to activate the plug-in.

Refer to Plug-ins for assistance with the `rabbitmq-plugins` command.

Configure JMS Applications to Use JMS Client for vFabric RabbitMQ

To enable JMS Client for vFabric RabbitMQ in a JEE container, you must install the supplied library files in the container and then define JMS resources in the container's naming system so that JMS clients can look them up. The methods for accomplishing these tasks are container-specific. Instructions for setting up the JMS Client for RabbitMQ in a vFabric tc Server runtime instance are included here. For other JEE containers, refer to the vendors' documentation.

Installing the JMS Client for RabbitMQ Libraries

The JMS Client for RabbitMQ requires the following libraries to be in the Java classpath when the JMS application executes:

- `rabbitmq-jms-`*`version`*`.jar`
- `amqp-client-`*`version`*`.jar`
- `geronimo-jms_1.1_spec-1.1.1.jar`

The files are in the JMS Client for RabbitMQ distribution. The `rabbitmq-jms-`*`version`*`.jar` is in the root directory of the expanded distribution archive, and the other two files are in the `dependencies` subdirectory.

With vFabric tc Server, you can choose one of two options for placing these JAR files in the classpath:

1. Copy the files into the `lib` directory of the tc Runtime instance, *`$CATALINA_HOME`*`/lib`.

   If you choose this option, any application executing on the tc Runtime instance can use JMS Client for RabbitMQ.

2. Copy the files into the `WEB-INF/lib` directory in the Web application.

   If you choose this option, the JMS Client for RabbitMQ libraries are only available to that Web application.

After you restart the tc Runtime instance or redeploy the Web application, the JAR files are available to the application on the classpath. The next step is to set up the JMS objects the application expects to find in the container.

Defining JMS Objects in JNDI

JMS clients typically use JNDI to look up a JMS ConnectionFactory and the JMS destinations where they send and receive messages. The `com.rabbitmq.jms.admin.RMQObjectFactory` class is an object factory that creates these objects in the container and registers them in JNDI. It can create the following types:

- `javax.jms.ConnectionFactory`
- `javax.jms.QueueConnectionFactory`
- `javax.jms.TopicConnectionFactory`
- `javax.jms.Topic`
- `javax.jms.Queue`

The objects it creates implement the corresponding `javax.jms` interfaces for RabbitMQ.

To configure JMS objects in containers like vFabric tc Server and Apache Tomcat, you add a `<Resource>` element in a configuration file. Here is an example of an entry for a JMS ConnectionFactory:

```
<Resource
          name="jms/ConnectionFactory"
          type="javax.jms.ConnectionFactory"
          factory="com.rabbitmq.jms.admin.RMQObjectFactory"
          username="guest"
          password="guest"
          virtualHost="/"
          host="localhost"
          threadsPerConnection="2"/>
```

Example entry for a JMS Topic:

```
<Resource
          name="jms/Topic"
          type="javax.jms.Topic"
```

```
          factory="com.rabbitmq.jms.admin.RMQObjectFactory"
          destinationName="myTopic"/>
```

Example entry for a JMS Queue:

```
<Resource
          name="jms/Queue"
          type="javax.jms.Queue"
          factory="com.rabbitmq.jms.admin.RMQObjectFactory"
          destinationName="myQueue"/>
```

In each example, the value of the `name` attribute is the JNDI name the client looks up. The JNDI name must match the name the application searches for to succeed. The `type` attribute specifies the type of object that is created, and the `factory` attribute specifies that the `com.rabbitmq.jms.admin.RMQObjectFactory` class is the factory class that creates the object. The remaining attributes are used to configure the object and are described below.

You can create the resource in the server globally or for an individual application, depending on where you add the `<Resource>` element. To configure a global resource, so that all applications can access it, add the entry to the `<GlobalNamingResources>` element in the `$CATALINA_BASE/conf/server.xml` file. To create it for a single Web application, add it to the `<Context>` element in the the `META-INF/context.xml` file in the Web application. If you add the resource to the `<GlobalNamingResources>` element, be sure you place the JMS Client for RabbitMQ library files in the *`$CATALINA`*`/lib` directory.

The attributes for the `jms/ConnectionFactory` resource are described in the following table.

Attribute	Description
`name`	Name to look up in JNDI, `jms/ConnectionFactory`.
`type`	Name of the JMS interface the object implements, usually `javax.jms.ConnectionFactory`. Other choices are `javax.jms.QueueConnectionFactory` and `javax.jms.TopicConnectionFactory`. You can also use the name of the implementation class for RabbitMQ, `com.rabbitmq.jms.admin.RMQConnectionFactory`.
`factory`	JMS Client for RabbitMQ ObjectFactory class, always `com.rabbitmq.jms.admin.RMQObjectFactory`.
`username`	Name to use to authenticate a connection with the RabbitMQ broker. The default is "guest".
`password`	Password to use to authenticate a connection with the RabbitMQ broker. The default is "guest".
`virtualHost`	RabbitMQ virtual host within which the application will operate. The default is "/".
`host`	Host on which RabbitMQ is running. The default is "localhost".
`port`	RabbitMQ listen port. The default is "5672".
`threadsPerConnection`	Number of threads each connection executor will have. The default is "2".
`threadPrefix`	Thread prefix for threads created by the executor. If not specified, the value "`Rabbit JMS Connection[`*`RabbitMQ-connection-address`*`]-`" is used.

The `jms/Topic` and `jms/Queue` resource types have the attributes described in the following table:

Attribute	Description
`name`	Name to look up in JNDI, `jms/ConnectionFactory`.

Attribute	Description
`type`	Name of the JMS interface the object implements, usually `javax.jms.ConnectionFactory`. Other choices are`javax.jms.QueueConnectionFactory` and `javax.jms.TopicConnectionFactory`. You can also use the name of the implementation class for RabbitMQ, `com.rabbitmq.jms.admin.RMQConnectionFactory`.
`factory`	JMS Client for RabbitMQ ObjectFactory class, always `com.rabbitmq.jms.admin.RMQObjectFactory`.
`destinationName`	Name of the JMS destination.

Configuring Logging for the JMS Client

Beginning with release 1.0.1, JMS Client for RabbitMQ logs messages using SLF4J (Simple Logging Façade for Java). SLF4J delegates to a logging framework, such as Apache log4j or JDK 1.4 logging (`java.util.logging`). If no other logging framework is enabled, SLF4J defaults to a built-in, no-op, logger. See the SLF4J documentation for a list of the logging frameworks SLF4J supports.

The target logging framework is configured at deployment time by adding an SLF4J binding for the framework to the classpath. For example, the log4j SLF4J binding is in the `slf4j-log4j12-`*`version`*`.jar` file, which is a part of the SLF4J distribution. To direct JMS client messages to log4j, for example, add the following JARs to the classpath:

- `slf4j-api-1.7.5.jar`
- `slf4j-log4j12-1.7.5.jar`
- `log4j.jar`

The SLF4J API is backwards compatible, so you can use use any version of SLF4J. Version 1.7.5 or higher is recommended. The SLF4J API and bindings, however, must be from the same SLF4J version.

No additional SLF4J configuration is required, once the API and binding JAR files are in the classpath. However, the target framework may have configuration files or command-line options. Refer to the documentation for the target logging framework for configuration details.

What's Next?

For details of vFabric RabbitMQ's implementation of the JMS API, see Chapter 4, *vFabric RabbitMQ Implementation of JMS API*.

4. vFabric RabbitMQ Implementation of JMS API

This section annotates the JMS Client for vFabric RabbitMQ implementation of the JMS 1.1 API.

You can download the JMS 1.1 specification and API documentation from the Oracle Technology Network Web site.

Subtopics

Connection Factory Interfaces

ConnectionFactory

`Connection CreateConnection()`	Supported
`Connection CreateConnection(java.lang.String userName, java.lang.String password)`	Supported

QueueConnectionFactory

`QueueConnection CreateQueueConnection()`	Supported
`QueueConnection CreateQueueConnection(java.lang.String userName, java.lang.String password)`	Supported

TopicConnectionFactory

`TopicConnection CreateTopicConnection()`	Supported
`TopicConnection CreateTopicConnection(java.lang.String userName, java.lang.String password)`	Supported

XAQueueConnectionFactory

`XAQueueConnection CreateXAQueueConnection()`	Supported

`XAQueueConnection CreateXAQueueConnection(java.lang.String *userName*, java.lang.String *password*)`	Supported

XATopicConnectionFactory

`XATopicConnection CreateXATopicConnection()`	Supported
`XATopicConnection CreateXATopicConnection(java.lang.String *userName*, java.lang.String *password*)`	Supported

Server Session Interfaces

The JMS for RabbitMQ client does not support server sessions.

ServerSessionPool

`ServerSession getServerSession()`	Not supported

ServerSession

`Session getSession()`	Not supported
`void start()`	Not supported

Connection Interfaces

Connection

`Session createSession(boolean *transacted*, int *acknowledgeMode*)`	Supported
`java.lang.String getClientID()`	Supported
`void setClientID(java.lang.String *clientID*)`	Supported
`ConnectionMetaData getMetaData()`	Not yet implemented
`ExceptionListener getExceptionListener()`	Supported
`void setExceptionListener(ExceptionListener *listener*)`	Supported
`void start()`	Supported
`void stop()`	Supported
`void close()`	Supported
`ConnectionConsumer createConnectionConsumer(Destination *destination*, java.lang.String *messageSelector*, ServerSessionPool *sessionPool*,`	Not supported

`int maxMessages)`	
`ConnectionConsumer createDurableConnectionConsumer(Topic topic, java.lang.String subscriptionName, java.lang.String messageSelector, ServerSessionPool sessionPool, int maxMessages)`	Not supported

QueueConnection

`QueueSession createQueueSession(boolean transacted, int acknowledgeMode)`	Supported
`ConnectionConsumer createConnectionConsumer(Queue queue, java.lang.String messageSelector, ServerSessionPool sessionPool, int maxMessages)`	Not supported

TopicConnection

`TopicSession createTopicSession(boolean transacted, int acknowledgeMode)`	Supported
`ConnectionConsumer createConnectionConsumer(Topic topic, java.lang.String messageSelector, ServerSessionPool sessionPool, int maxMessages)`	Not supported
`ConnectionConsumer createDurableConnectionConsumer(Topic topic, java.lang.String subscriptionName, java.lang.String messageSelector, ServerSessionPool sessionPool, int maxMessages)`	Not supported

XAConnection

`XASession createXASession()`	Not yet implemented
`Session createSession(boolean transacted, int acknowledgeMode)`	Not yet implemented

XAQueueConnection

`XAQueueSession createXAQueueSession()`	Not yet implemented
`QueueSession createQueueSession(boolean transacted, int acknowledgeMode)`	Not yet implemented

XATopicConnection

`XATopicSession createXATopicSession()`	Not yet implemented
`TopicSession createTopicSession(boolean transacted,`	Not yet implemented

`int acknowledgeMode)`	

Session Interfaces

Session

`BytesMessage createBytesMessage()`	Supported
`MapMessage createMapMessage()`	Supported
`Message createMessage()`	Supported
`ObjectMessage createObjectMessage()`	Supported
`ObjectMessage createObjectMessage(java.io.Serializable object)`	Supported
`StreamMessage createStreamMessage()`	Supported
`TextMessage createTextMessage()`	Supported
`TextMessage createTextMessage(java.lang.String text)`	Supported
`boolean getTransacted()`	Supported
`int getAcknowledgeMode()`	Supported
`void commit()`	Supported
`void rollback()`	Supported
`void close()`	Supported
`void recover()`	Supported
`MessageListener getMessageListener()`	Supported
`void setMessageListener(MessageListener listener)`	Supported
`void run()`	Not supported
`MessageProducer createProducer(Destination destination)`	Supported
`MessageConsumer createConsumer(Destination destination)`	Supported
`MessageConsumer createConsumer(Destination destination, java.lang.String messageSelector)`	Not yet implemented
`MessageConsumer createConsumer(Destination destination, java.lang.String messageSelector, boolean NoLocal)`	Supported without NoLocal
`Queue createQueue(java.lang.String queueName)`	Supported

`Topic createTopic(java.lang.String topicName)`	Supported
`TopicSubscriber createDurableSubscriber(Topic topic, java.lang.String name)`	Supported
`TopicSubscriber createDurableSubscriber(Topic topic, java.lang.String name, java.lang.String messageSelector, boolean noLocal)`	Supported without NoLocal
`QueueBrowser createBrowser(Queue queue)`	Not yet implemented
`QueueBrowser createBrowser(Queue queue, java.lang.String messageSelector)`	Not yet implemented
`TemporaryQueue createTemporaryQueue()`	Supported
`TemporaryTopic createTemporaryTopic()`	Supported
`void unsubscribe(java.lang.String name)`	Supported for durable subscriptions only

TopicSession

`Topic createTopic(java.lang.String topicName)`	Supported
`TopicSubscriber createSubscriber(Topic topic, java.lang.String messageSelector, boolean noLocal)`	NoLocal is not supported
`TopicSubscriber createSubscriber(Topic topic)`	Supported
`TopicSubscriber createDurableSubscriber(Topic topic, java.lang.String name)`	Supported

QueueSession

`Queue createQueue(java.lang.String queueName))`	Supported
`QueueReceiver createReceiver(Queue queue)`	Supported
`QueueReceiver createReceiver(Queue queue, java.lang.String messageSelector)`	Not yet implemented
`QueueSender createSender(Queue queue)`	Supported
`QueueBrowser createBrowser(Queue queue)`	Supported
`QueueBrowser createBrowser(Queue queue, java.lang.String messageSelector)`	Supported
`TemporaryQueue createTemporaryQueue()`	Supported

XAQueueSession

`QueueSession getQueueSession()`	Not yet implemented

XASession

`Session getSession()`	Not yet implemented
`XAResource getXAResource()`	Not yet implemented
`boolean getTransacted()`	Not yet implemented
`void commit()`	Not yet implemented
`void rollback()`	Not yet implemented

XATopicSession

`TopicSession getTopicSession()`	Not yet implemented

Consumer and Producer Interfaces

ConnectionConsumer

`ServerSessionPool getServerSessionPool()`	Not supported
`void close()`	Not Supported

MessageProducer

`void setDisableMessageID(boolean value)`	Ignored.
`boolean getDisableMessageID()`	Ignored.
`void setDisableMessageTimestamp(boolean value)`	Ignored.
`boolean getDisableMessageTimestamp()`	Ignored.
`void setDeliveryMode(int deliveryMode)`	Supported
`int getDeliveryMode()`	Supported
`void setPriority(int defaultPriority)`	Supported
`int getPriority()`	Supported
`void setTimeToLive(long timeToLive)`	Supported
`long getTimeToLive()`	Supported
`Destination getDestination()`	Supported

`void close()`	Supported
`void send(Message message)`	Supported
`void send(Message message, int deliveryMode, int priority, long timeToLive)`	Supported
`void send(Destination destination, Message message)`	Supported
`void send(Destination destination, Message message, int deliveryMode, int priority, long timeToLive)`	Supported

QueueSender

`Queue getQueue()`	Supported
`void send(Message message)`	Supported
`void send(Message message, int deliveryMode, int priority, long timeToLive)`	Supported
`void send(Queue queue, Message message)`	Supported
`void send(Queue queue, Message message, int deliveryMode, int priority, long timeToLive)`	Supported

TopicPublisher

`Topic getTopic()`	Supported
`void publish(Message message)`	Supported
`void publish(Message message, int deliveryMode, int priority, long timeToLive)`	Supported
`void publish(Topic topic, Message message)`	Supported
`void publish(Topic topic, Message message, int deliveryMode, int priority, long timeToLive)`	Supported

Message Interfaces

Message

`java.lang.String getJMSMessageID()`	Supported
`void setJMSMessageID(java.lang.String id)`	Supported
`long getJMSTimestamp()`	Supported
`void setJMSTimestamp(long timestamp)`	Supported
`byte[] getJMSCorrelationIDAsBytes()`	Supported
`void setJMSCorrelationIDAsBytes(byte[] correlationID)`	Supported
`void setJMSCorrelationID(java.lang.String correlationID)`	Supported
`java.lang.String getJMSCorrelationID()`	Supported
`Destination getJMSReplyTo()`	Supported
`void setJMSReplyTo(Destination replyTo)`	Supported
`Destination getJMSDestination()`	Supported
`void setJMSDestination(Destination destination)`	Supported
`int getJMSDeliveryMode()`	Supported
`void setJMSDeliveryMode(int deliveryMode)`	Supported
`boolean getJMSRedelivered()`	Supported
`void setJMSRedelivered(boolean redelivered)`	Supported
`java.lang.String getJMSType()`	Supported
`void setJMSType(java.lang.String type)`	Supported
`long getJMSExpiration()`	Supported
`void setJMSExpiration(long expiration)`	Supported
`int getJMSPriority()`	Supported
`void setJMSPriority(int priority)`	Supported
`void clearProperties()`	Supported
`boolean propertyExists(java.lang.String name)`	Supported

`boolean getBooleanProperty(java.lang.String name)`	Supported
`byte getByteProperty(java.lang.String name)`	Supported
`short getShortProperty(java.lang.String name)`	Supported
`int getIntProperty(java.lang.String name)`	Supported
`long getLongProperty(java.lang.String name)`	Supported
`float getFloatProperty(java.lang.String name)`	Supported
`double getDoubleProperty(java.lang.String name)`	Supported
`java.lang.String getStringProperty(java.lang.String name)`	Supported
`java.lang.Object getObjectProperty(java.lang.String name)`	Supported
`java.util.Enumeration getPropertyNames()`	Supported
`void setBooleanProperty(java.lang.String name, boolean value)`	Supported
`void setShortProperty(java.lang.String name, short value)`	Supported
`void setIntProperty(java.lang.String name, int value)`	Supported
`void setLongProperty(java.lang.String name, long value)`	Supported
`void setFloatProperty(java.lang.String name, float value)`	Supported
`void setDoubleProperty(java.lang.String name, double value)`	Supported
`void setStringProperty(java.lang.String name, java.lang.String value)`	Supported
`void setObjectProperty(java.lang.String name, java.lang.Object value)`	Supported
`void acknowledge()`	Supported
`void clearBody()`	Supported

BytesMessage

`long getBodyLength()`	Supported
`boolean readBoolean()`	Supported

`byte readByte()`	Supported
`int readUnsignedByte()`	Supported
`short readShort()`	Supported
`int readUnsignedShort()`	Supported
`char readChar()`	Supported
`int readInt()`	Supported
`long readLong()`	Supported
`float readFloat()`	Supported
`double readDouble()`	Supported
`java.lang.String readUTF()`	Supported
`int readBytes(byte[] value)`	Supported
`int readBytes(byte[] value,` `int length)`	Supported
`void writeBoolean(boolean value)`	Supported
`void writeByte(byte value)`	Supported
`void writeShort(short value)`	Supported
`void writeChar(char value)`	Supported
`void writeInt(int value)`	Supported
`void writeLong(long value)`	Supported
`void writeFloat(float value)`	Supported
`void writeDouble(double value)`	Supported
`void writeUTF(java.lang.String value)`	Supported
`void writeBytes(byte[] value)`	Supported

`void writeBytes(byte[] value,`
`int offset,`
`int length)`
	Supported

`void writeObject(java.lang.Object value)`	Supported
`void reset()`	Supported

MapMessage

`boolean getBoolean(java.lang.String name)`	Supported
`byte getByte(java.lang.String name)`	Supported
`short getShort(java.lang.String name)`	Supported
`char getChar(java.lang.String name)`	Supported
`int getInt(java.lang.String name)`	Supported
`long getLong(java.lang.String name)`	Supported
`float getFloat(java.lang.String name)`	Supported
`double getDouble(java.lang.String name)`	Supported
`java.lang.String getString(java.lang.String name)`	Supported
`byte[] getBytes(java.lang.String name)`	Supported
`java.lang.Object getObject(java.lang.String name)`	Supported
`java.util.Enumeration getMapNames()`	Supported
`void setBoolean(java.lang.String name, boolean value)`	Supported
`void setByte(java.lang.String name, byte value)`	Supported
`void setShort(java.lang.String name, short value)`	Supported
`void setChar(java.lang.String name, char value)`	Supported
`void setInt(java.lang.String name, int value)`	Supported
`void setLong(java.lang.String name, long value)`	Supported
`void setFloat(java.lang.String name, float value)`	Supported
`void setDouble(java.lang.String name, double value)`	Supported
`void setString(java.lang.String name, java.lang.String value)`	Supported
`void setBytes(java.lang.String name,`	Supported

byte[] *value*)	
void setBytes(java.lang.String *name*, byte[] *value*, int *offset*, int *length*)	Supported
void setObject(java.lang.String *name*, java.lang.Object *value*)	Supported
boolean itemExists(java.lang.String *name*)	Supported

ObjectMessage

void setObject(java.io.Serializable *object*)	Supported
java.io.Serializable getObject()	Supported

StreamMessage

boolean readBoolean()	Supported
byte readByte()	Supported
short readShort()	Supported
char readChar()	Supported
int readInt()	Supported
long readLong()	Supported
float readFloat()	Supported
double readDouble()	Supported
java.lang.String readString()	Supported
int readBytes(byte[] *value*)	Supported
java.lang.Object readObject()	Supported
void writeBoolean(boolean *value*)	Supported
oid writeByte(byte *value*)	Supported
void writeShort(short *value*)	Supported
void writeChar(char *value*)	Supported
void writeInt(int *value*)	Supported

`void writeLong(long value)`	Supported
`void writeFloat(float value)`	Supported
`void writeDouble(double value)`	Supported
`void writeString(java.lang.String value)`	Supported
`void writeBytes(byte[] value)`	Supported
`void writeBytes(byte[] value, int offset, int length)`	Supported
`void writeObject(java.lang.Object value)`	Supported
`void reset()`	Supported

TextMessage

`void setText(java.lang.String string)`	Supported
`java.lang.String getText()`	Supported

Message Consumer Interfaces

MessageConsumer

`java.lang.String getMessageSelector()`	Supported
`MessageListener getMessageListener()`	Supported
`void setMessageListener(MessageListener listener)`	Supported
`Message receive()`	Supported
`Message receive(long timeout)`	Supported
`Message receiveNoWait()`	Supported
`void close()`	Supported

QueueReceiver

`Queue getQueue()`	Supported

TopicSubscriber

`Topic getTopic()`	Supported

`boolean getNoLocal()`	NoLocal is not supported

Destination Interfaces

Destination

(Has No Methods)

Queue

`java.lang.String getQueueName()`	Supported
`java.lang.String toString()`	Supported

TemporaryQueue

`void delete()`	Supported

Topic

`java.lang.String getTopicName()`	Supported
`java.lang.String toString()`	Supported

TemporaryTopic

`void delete()`	Supported

QueueBrowser

`Queue getQueue()`	Not yet implemented
`java.lang.String getMessageSelector()`	Not yet implemented
`java.util.Enumeration getEnumeration()`	Not yet implemented
`void close()`	Not yet implemented