Pivotal

PRODUCT DOCUMENTATION

Pivotal™ RabbitMQ®

Version 3.4

Using the JMS Client for Pivotal RabbitMQ

Rev: 01

Notice

Copyright

Contents

Contacting Pivotal

Current Pivotal Customers

Submit a ticket from the *Help & Support Page*.

Public Inquiries About Application Fabric Products

Email the appropriate group for the Application Fabric product:

- RabbitMQ@goPivotal.com
- tcServer@goPivotal.com
- WebServer@goPivotal.com

Chapter 1

About Using the JMS Client for Pivotal RabbitMQ

Using the JMS Client for Pivotal RabbitMQ describes using the JMS client with the Pivotal 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.

About JMS Client for Pivotal RabbitMQ

JMS Client for Pivotal RabbitMQ is a client library for Pivotal RabbitMQ. Pivotal 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).

Components of JMS Client for Pivotal RabbitMQ

The JMS Client for Pivotal 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 Pivotal 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.
- SSL and socket options for RabbitMQ connections are supported, but only using the (default) SSL connection protocols that the RabbitMQ client provides.
- 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 *Pivotal RabbitMQ Implementation of JMS API* for a detailed list of supported JMS APIs.

Installing and Configuring JMS Client for Pivotal RabbitMQ

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

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

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

Download and Install JMS Client for Pivotal RabbitMQ

1. From the *Pivotal RabbitMQ product page*, click **Downloads**.
2. Download the `rabbitmq-jms-package-`*`version`*`-client-and-plugin.tar.gz` or `rabbitmq-jms-package-`*`version`*`-client-and-plugin.zip` file.
3. 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 Pivotal RabbitMQ*.

Enable the Pivotal RabbitMQ Topic Selector Plug-in

The topic selector plug-in is included with Pivotal RabbitMQ. Like any RabbitMQ plugin, you need to enable the plug-in in order to use it.

1. Enable the plug-in using the `rabbitmq-plugins` command:

   ```
   $ rabbitmq-plugins enable rabbitmq_jms_topic_exchange
   ```

   Note that the plug-in name, `rabbit_jms_topic_exchange`, is not the same as the file name.
2. Restart the RabbitMQ server to activate the plug-in.

Configure JMS Applications to Use JMS Client for Pivotal RabbitMQ

To enable JMS Client for Pivotal 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 Pivotal tc Server runtime instance are included here. For other JEE containers, refer to the vendors' documentation.

JMS and AMQP Destination Interoperability

Configuring Logging for the JMS Client

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 Pivotal 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

The `RMQConnectionFactory` object has properties/attributes that can be set on `Resource` or Spring bean definitions. These include the `host` and `virtualHost` values of the connections created.

To configure JMS objects in containers like Pivotal tc Server and Apache Tomcat, you add a `<Resource>` element in a configuration file. For example, a JNDI `ConnectionFactory` `<Resource/>` could be defined as follows:

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

Here is the equivalent Spring bean example:

```
<bean id="jmsConnectionFactory" class="com.rabbitmq.jms.admin.RMQConnectionFactory" >
  <property name="username" value="guest" />
  <property name="password" value="guest" />
  <property name="virtualHost" value="/" />
  <property name="host" value="localhost" />
  <property name="ssl" value="true" />
</bean>
```

The following table lists all of the attributes/properties that are available.

Attribute/Property	JNDI only?	Description
`name`	JNDI only	Name in JNDI.

Attribute/Property	JNDI only?	Description
`type`	JNDI only	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 (common) implementation class, `com.rabbitmq.jms.admin.RMQConnectionFactory`.
`factory`	JNDI only	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 port used for connections. The default is "5672" unless this is an SSL connection, in which case the default is "5671".
`ssl`		Whether to use an SSL connection to RabbitMQ. The default is "false".
`uri`		The AMQP URI string used to establish a RabbitMQ connection. The value can encode the `host`, `port`, `userid`, `password` and `virtualHost` in a single string. Both ‘amqp’ and ‘amqps’ schemes are accepted. See the *AMQP URI specification*on the public RabbitMQ site for details. Note: this property sets other properties and the set order is unspecified.

JMS and AMQP Destination Interoperability

An interoperability feature, introduced in Pivotal JMS Client for RabbitMQ 1.1, allows you to define JMS 'amqp' destinations that read and/or write to non-JMS RabbitMQ resources

A JMS destination can be defined so that a JMS application can send `Message`s to a predefined RabbitMQ 'destination' (exchange/routing key) using the JMS API in the normal way. The messages are written "in the clear," which means that any AMQP client can read them without having to understand the internal format of Java JMS messages. Only `BytesMessage`s and `TextMessage`s can be written in this way.

Similarly, a JMS destination can be defined that reads messages from a a predefined RabbitMQ queue. A JMS application can then read these messages using the JMS API. JMS Client for RabbitMQ packs them up into JMS `Message`s automatically. Messages read in this way are, by default, `BytesMessage`s, but individual messages can be marked `TextMessage` (by adding an AMQP message property called "JMSType" whose value is "TextMessage"), which will interpret the byte-array payload as a UTF8 encoded `String` and return them as `TextMessage`s.

A single 'amqp' destination can be defined for both reading and writing.

When messages are sent to an 'amqp' Destination, JMS message properties are mapped onto AMQP headers and properties as appropriate. For example, the `JMSPriority` property converts to the `priority` property for the AMQP message. (It is also set as a header with the name "`JMSPriority`".) User-defined properties are set as named message header values, provided they are `boolean`, numeric or `String` types.

When reading from an 'amqp' Destination, values are mapped back to JMS message properties, except that any explicit JMS property set as a message header overrides the natural AMQP header value, unless this would misrepresent the message. For example, `JMSDeliveryMode` cannot be overridden in this way.

JMS 'amqp' `RMQDestination` Constructor

The `com.rabbitmq.jms.admin` package contains the `RMQDestination` class, which implements `Destination` in the JMS interface. This is extended with a new constructor:

```
public RMQDestination(String destinationName, String amqpExchangeName,
                      String amqpRoutingKey, String amqpQueueName);
```

Description

Creates a destination for JMS for RabbitMQ mapped onto an AMQP resource.

Parameters

- `destinationName` – the name of the queue destination
- `amqpExchangeName` – the exchange name for the map resource
- `amqpRoutingKey` – the routing key for the mapped resource
- `amqpQueueName` – the queue name of the mapped resource

Applications that declare destinations in this way can use them directly, or store them in a JNDI provider for JMS applications to retrieve. Such destinations are non-temporary, queue destinations.

JMS AMQP Destination XML Definitions

The `RMQDestination` object has the following new instance fields:

- `amqp` – *boolean*, indicates if this is an AMQP destination (if **true**); the default is **false**.
- `amqpExchangeName` – *String*, the AMQP exchange name to use when sending messages to this destination, if `amqp` is **true**; the default is **null**.
- `amqpRoutingKey` – *String*, the AMQP routing key to use when sending messages to this destination, if `amqp` is **true**; the default is **null**.
- `amqpQueueName` – *String*, the AMQP queue name to use when reading messages from this destination, if `amqp` is **true**; the default is **null**.

There are getters and setters for these fields, which means that a JNDI `<Resource/>` definition or a Spring bean definition can use them, for example:

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

This is the equivalent Spring bean example:

```
  <bean id="jmsDestination" class="com.rabbitmq.jms.admin.RMQDestination" >
    <property name="destinationName" value="myQueue" />
    <property name="amqp"            value="true" />
    <property name="amqpQueueName"   value="rabbitQueueName" />
  </bean>
```

Following is a *complete* list of the attributes/properties that are available:

Attribute/ Property Name	JNDI Only?	Description
`name`	JNDI only	Name in JNDI.
`type`	JNDI only	Name of the JMS interface the object implements, usually `javax.jms.Queue`. Other choices are `javax.jms.Topic` and `javax.jms.Destination`. You can also use the name of the (common) implementation class, `com.rabbitmq.jms.admin.RMQDestination`.
`factory`	JNDI only	JMS Client for RabbitMQ `ObjectFactory` class, always `com.rabbitmq.jms.admin.RMQObjectFactory`.
`amqp`		"**true**" means this is an 'amqp' destination. Default "**false**".
`ampqExchangeName`		Name of the AMQP exchange to publish messages to when an 'amqp' destination. This exchange must exist when messages are published.
`amqpRoutingKey`		The routing key to use when publishing messages when an 'amqp' destination.
`amqpQueueName`		Name of the AMQP queue to receive messages from when an 'amqp' destination. This queue must exist when messages are received.
`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 Pivotal RabbitMQ's implementation of the JMS API, see *Pivotal RabbitMQ Implementation of JMS API*.

Pivotal RabbitMQ Implementation of JMS API

This section annotates the JMS Client for Pivotal 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*.

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()`	Not supported
`XAQueueConnection CreateXAQueueConnection(java.lang.String userName, java.lang.String password)`	Not supported

XATopicConnectionFactory

`XATopicConnection CreateXATopicConnection()`	Not supported
`XATopicConnection CreateXATopicConnection(java.lang.String userName, java.lang.String password)`	Not supported

Server Session Interfaces

The JMS for RabbitMQ client does not support server sessions.

ServerSessionPool

ServerSession

ServerSessionPool

`ServerSession getServerSession()`	Not supported

ServerSession

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

Connection Interfaces

Connection

QueueConnection

TopicConnection

XAConnection

XAQueueConnection

XATopicConnection

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, int maxMessages)`	Not supported
`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, int acknowledgeMode)`	Not yet implemented

Session Interfaces

Session

TopicSession

QueueSession

XAQueueSession

XASession

XATopicSession

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 implemented for non-empty messageSelector
`MessageConsumer createConsumer(Destination destination, java.lang.String messageSelector, boolean NoLocal)`	Not implemented for non-empty messageSelector, and noLocal accepted but ignored
`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

MessageProducer

QueueSender

TopicPublisher

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

BytesMessage

MapMessage

ObjectMessage

StreamMessage

TextMessage

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, byte[] value)`	Supported
`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

QueueReceiver

TopicSubscriber

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

Queue

TemporaryQueue

Topic

TemporaryTopic

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

See *QueueBrowser support* for implementation details.

`Queue getQueue()`	Supported
`java.lang.String getMessageSelector()`	Supported
`java.util.Enumeration getEnumeration()`	Supported
`void close()`	Supported

API Implementation Details

This section provides additional implementation details for specific JMS API classes in JMS Client for RabbitMQ.

Deviations from the specification are implemented to support common acknowledgement behaviours.

`QueueBrowser` support

Overview of queue browsers

The JMS API includes objects and methods to browse an existing queue destination, reading its messages *without* removing them from the queue. Topic destinations cannot be browsed in this manner.

A `QueueBrowser` can be created from a (queue) `Destination`, with or without a selector expression. The browser has a `getEnumeration()` method, which returns a Java `Enumeration` of `Message`s copied from the queue.

If no selector is supplied, then all messages in the queue appear in the `Enumeration`. If a selector is supplied, then only those messages that satisfy the selector appear.

Implementation

The destination queue is read when the `getEnumeration()` method is called. A *snapshot* is taken of the messages in the queue; and the selector expression, if one is supplied, is used at this time to discard messages that do not match.

The message copies may now be read using the `Enumeration` interface (`nextElement()` and `hasMoreElements()`).

The selector expression and the destination queue of the `QueueBrowser` may not be adjusted after the `QueueBrowser` is created.

An `Enumeration` cannot be "reset", but the `getEnumeration()` method may be re-issued, taking a *new* snapshot from the queue each time.

The contents of an `Enumeration` survive session and/or connection close, but a `QueueBrowser` may not be used after the session that created it has closed. `QueueBrowser.close()` has no effect.

Implementation Details

Which messages are included

Messages that arrive, expire, are re-queued, or are removed after the `getEnumeration()` call have no effect on the contents of the`Enumeration` it produced. If the messages in the queue change *while the* `Enumeration` *is being built*, they may or may not be included. In particular, if messages from the queue are simultaneously read by another client (or session), they may or may not appear in the`Enumeration`.

Message copies do not "expire" from an `Enumeration`.

Order of messages

If other client sessions read from a queue that is being browsed, then it is possible that some messages may subsequently be received out of order.

Message order will not be disturbed if no other client sessions read the queue at the same time.

Memory usage

When a message is read from the `Enumeration` (with `nextElement()`), then no reference to it is retained in the Java Client. This means the storage it occupies in the client is eligible for release (by garbage collection) if no other references are retained. Retaining an `Enumeration` will retain the storage for all message copies that remain in it.

If the queue has many messages -- or the messages it contains are very large -- then a `getEnumeration()` method call may consume a large amount of memory in a very short time. This remains true even if only a few messages are selected. There is currently limited protection against `OutOfMemoryError` conditions that may arise because of this. See the next section.

Setting a maximum number of messages to browse

Each connection is created with a limit on the number of messages that are examined by a `QueueBrowser`. The limit is set on the `RMQConnectionFactory` by `RMQConnectionFactory.setQueueBrowserReadMax(int)` and is passed to each `Connection` subsequently created by`ConnectionFactory.createConnection()`.

The limit is an integer that, if positive, stops the queue browser from reading more than this number of messages when building an enumeration. If it is zero or negative, it is interpreted as imposing no limit on the browser, and all of the messages on the queue are scanned.

The default limit for a factory is determined by the `rabbit.jms.queueBrowserReadMax` system property, if set, and the value is specified as `0` if this property is not set or is not an integer.

If a `RMQConnectionFactory` value is obtained from a JNDI provider, then the limit set when the factory object was created is preserved.

RJMS Release support

Support for `QueueBrowser`s is introduced in JMS Client for RabbitMQ 1.2.0. Prior to that release, calling `Session.createBrowser(Queue queue[, String selector])` resulted in an `UnsupportedOperationException`.

Group and individual acknowledgement

Prior to JMS Client for RabbitMQ 1.2.0, in client acknowledgement mode (`Session.CLIENT_ACKNOWLEDGE`), acknowledging any message from an open session would acknowledge *every* unacknowledged message of that session, whether they were received before or after the message being acknowledged.

Currently, the behaviour of `Session.CLIENT_ACKNOWLEDGE` mode is modified so that, when calling `msg.acknowledge()`, only the message `msg` *and all* previously received *unacknowledged messages on that session* are acknowledged. Messages received *after* `msg` was received are not affected. This is a form of *group acknowledgement*, which differs slightly from the JMS 1.1 specification but is likely to be more useful, and is compatible with the vast majority of uses of the existing acknowledge function.

For even finer control, a new acknowledgement mode may be set when creating a session, called `RMQSession.CLIENT_INDIVIDUAL_ACKNOWLEDGE`.

A session created with this acknowledgement mode will mean that messages received on that session will be acknowledged individually. That is, the call `msg.acknowledge()` will acknowledge only the message `msg` and not affect any other messages of that session.

The acknowledgement mode `RMQSession.CLIENT_INDIVIDUAL_ACKNOWLEDGE` is equivalent to `Session.CLIENT_ACKNOWLEDGE` in all other respects. In particular the `getAcknowledgeMode()` method returns `Session.CLIENT_ACKNOWLEDGE` even if `RMQSession.CLIENT_INDIVIDUAL_ACKNOWLEDGE` has been set.

Arbitrary `Message` support

Any instance of a class that implements the `javax.jms.Message` interface can be *sent* by an RJMS message producer.

All properties of the message required by `send()` are correctly interpreted except that the `JMSReplyTo` header and objects (as property values or the body of an `ObjectMessage`) that cannot be deserialized are ignored.

The implementation extracts the properties and body from the `Message` instance using interface methods and recreates it as a message of the right (`RMQMessage`) type (`BytesMessage`, `MapMessage`, `ObjectMessage`, `TextMessage`, or `StreamMessage`) before sending it. This means that there is some performance loss due to the copying; but in the normal case, when the message is an instance of`com.rabbitmq.jms.client.RMQMessage`, no copying is done.