Configuring iWay Service Manager for RabbitMQ Using Java Message Service

RabbitMQ is open source message broker software that implements the Advanced Message Queuing Protocol (AMQP). The RabbitMQ server is written in the Erlang programming language and is built on the Open Telecom Platform (OTP) framework for clustering and failover. Client libraries to interface with the broker are available for all major programming languages.

This topic describes how to configure iWay Service Manager (iSM) for RabbitMQ using Java Message Service (JMS).

Prerequisites

Before continuing, ensure that you review the prerequisites that are described in this section.

Download From Pivotal

- Pivotal RabbitMQ Server Version 3.5.6
- JMS Package (Client and Plugin) Version 1.4.6 for Pivotal RabbitMQ Server Version 3.5.6

This software is available for download from:

https://network.pivotal.io/products/pivotal-rabbitmq

You will need to register prior to downloading this software.

You must have the correct version of the JMS package for the version of the RabbitMQ server that you are using (for example, version 3.5.6). JMS is only available for the commercial version of RabbitMQ.

To install and run the RabbitMQ server, you must first install the Erlang programming language, which can be downloaded from:

http://www.erlang.org/downloads

For example, you can install Open Telecom Platform (OTP) Version 18.2.1 (*otp_win64_18.2.1.exe*).

Download From Oracle

- File System Service Provider Version 1.2 Beta 3 (*fscontext-1_2-beta3.zip*)

This software is available for download from:

http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-java-plat-419418.html

This is a File System Context implementation of JNDI, which is the simplest version of JNDI to set up. It allows the JMS components to work with RabbitMQ connections and destinations.

Installing RabbitMQ Server

To install the RabbitMQ server:

1. Install the Erlang programming language from the package you downloaded, as described in the *Prerequisites* section.

2. Install the RabbitMQ server as described in:

   http://rabbitmq.docs.pivotal.io/35/topics/install-getstart.html#install-windows

   Note: After uninstalling one instance of the RabbitMQ server and installing another instance, you may be unable to start the RabbitMQ server. On Windows, the Event Viewer shows the following message:

   ```
   RabbitMQ: Erlang machine stopped instantly (distribution name conflict?). The service is not restarted as OnFail is set to ignore.
   ```

   As a workaround, delete the AppData directory (for example, C:\Temp*AppData*\Roaming\RabbitMQ), as described in:

   https://groups.google.com/forum/#!topic/rabbitmq-users/138RHzzsORU

3. Deploy the Management plugin as described in the RabbitMQ server installation documentation.

   The Management plugin starts the Management console, which informs you about what queues you have and how many messages have been delivered. The Management console can be accessed using the following URL:

   http://localhost:15672

4. Log in to the Management console using the following credentials:

   - **Username:** guest
   - **Password:** guest

Configuring iWay Service Manager

This section describes how to configure the required components in iWay Service Manager (iSM) for RabbitMQ.

JNDI Configuration

For fscontext JNDI, you must create a bindings file. This is a text file that specifies the classes to load for JMS connections and destinations, and describes the available destinations. The file must always be named using a *.bindings* extension.

You can find a discussion of the JNDI setup for RabbitMQ at:

http://kaazing.com/doc/jms/4.0/integration-jms/p_jms_integrate_rabbitmq.html.

However, your bindings file may contain a few differences as a result of version changes.

The following is a valid example that defines two queues (*RequestQ* and *ResponseQ*).

```
ConnectionFactory/ClassName=javax.jms.ConnectionFactory
ConnectionFactory/FactoryName=com.rabbitmq.jms.admin.RMQObjectFactory

ConnectionFactory/RefAddr/0/Content=jms/ConnectionFactory
ConnectionFactory/RefAddr/0/Type=name
ConnectionFactory/RefAddr/0/Encoding=String

ConnectionFactory/RefAddr/1/Content=javax.jms.ConnectionFactory
ConnectionFactory/RefAddr/1/Type=type
ConnectionFactory/RefAddr/1/Encoding=String

ConnectionFactory/RefAddr/2/Content=com.rabbitmq.jms.admin.RMQObjectFactory
ConnectionFactory/RefAddr/2/Type=factory
ConnectionFactory/RefAddr/2/Encoding=String

# Change this line accordingly if the broker is not at localhost
ConnectionFactory/RefAddr/3/Content=localhost
ConnectionFactory/RefAddr/3/Type=host
ConnectionFactory/RefAddr/3/Encoding=String

RequestQ/ClassName=javax.jms.Queue
RequestQ/FactoryName=com.rabbitmq.jms.admin.RMQObjectFactory

RequestQ/RefAddr/0/Content=jms/Queue
RequestQ/RefAddr/0/Type=name
RequestQ/RefAddr/0/Encoding=String

RequestQ/RefAddr/1/Content=javax.jms.Queue
RequestQ/RefAddr/1/Type=type
RequestQ/RefAddr/1/Encoding=String

RequestQ/RefAddr/2/Content=com.rabbitmq.jms.admin.RMQObjectFactory
RequestQ/RefAddr/2/Type=factory
RequestQ/RefAddr/2/Encoding=String

RequestQ/RefAddr/3/Content=RequestQ
RequestQ/RefAddr/3/Type=destinationName
RequestQ/RefAddr/3/Encoding=String

ResponseQ/ClassName=javax.jms.Queue
ResponseQ/FactoryName=com.rabbitmq.jms.admin.RMQObjectFactory

ResponseQ/RefAddr/0/Content=jms/Queue
ResponseQ/RefAddr/0/Type=name
ResponseQ/RefAddr/0/Encoding=String

ResponseQ/RefAddr/1/Content=javax.jms.Queue
ResponseQ/RefAddr/1/Type=type
ResponseQ/RefAddr/1/Encoding=String

ResponseQ/RefAddr/2/Content=com.rabbitmq.jms.admin.RMQObjectFactory
ResponseQ/RefAddr/2/Type=factory
ResponseQ/RefAddr/2/Encoding=String

ResponseQ/RefAddr/3/Content=ResponseQ
ResponseQ/RefAddr/3/Type=destinationName
ResponseQ/RefAddr/3/Encoding=String
```

Note: If you require other queues, you can copy from the above example. Ensure to change the value of *<JNDI queue name>/RefAddr/3/Content* from the original you copied, or else your new definition will point to the same queue in RabbitMQ as the original definition.

Save the bindings file (*.bindings*) in a convenient directory.

iSM Classpath Additions

1. From the JMS package (client and plugin) for RabbitMQ server that you downloaded from Pivotal, extract the following .jar files:

   - *rabbitmq-jms-1.4.6.jar*
   - *amqp-client-3.5.6.jar*
   - *geronimo-jms-1.1_spec-1.1.1.jar*

   You can save these .jar files outside of the iWay Service Manager (iSM) directory structure and add them to the iSM classpath using the iSM Administration Console. You can also copy these .jar files to your <*iway_home*>\lib directory. For example:

   C:\iway7\lib

2. From the *fscontext-1_2-beta3.zip* file that you downloaded from Oracle, extract the following .jar files:

   - *fscontext.jar*
   - *providerutils.jar*

   You can save these .jar files outside of the iSM directory structure and add them to the iSM classpath using the iSM Administration Console. You can also copy these .jar files to your <*iway_home*>\lib directory. For example:

   C:\iway7\lib

3. Restart iSM to make the new libraries available.

Configuring the Java Message Service Queue Emit Service

1. Create a process flow that includes the Java Message Service Queue (JMSQ) Emit Service (com.ibi.agents.XDJMSQEmitAgent).

   The following table lists and describes the parameters that you must configure for the JMSQ Emit Service.

Parameter	Value	Description
Receiver Name	RequestQ	As defined in the JNDI .bindings file.
Connection Factory	ConnectionFactory	As defined in the JNDI .bindings file.
JNDI Factory	com.sun.jndi.fscontext.RefFSContextFactory	This value is specified since fscontext is being used.

JNDI URL	file:/c:/working	This is a file URL that points to the directory where the .bindings file was saved. Note that this value does not point directly to the file, but just to the directory. **Note:** This parameter must use a file URL (for example, c:/…). If you do not want to use a file URL, see the JNDI file system documentation (*jndi-fs.html*) that is included with the *fscontext-1_2-beta3.zip* package.
User	guest	This is the default user ID being used unless you changed the RabbitMQ security settings.
Password	guest	This is the default password being used unless you changed the RabbitMQ security settings.
Messaging Style	Queue	
Output Message Type	Dynamic	
JMSType	TextMessage	

2. Deploy this process flow to a simple channel.

   A simple channel that uses a File Listener is a good strategy to use for testing purposes.

3. Start the channel and drop a simple message that can be picked up by the File Listener.

   You can verify that a message has reached the queue using the RabbitMQ Management console.

Configuring the Java Message Service Queue Listener

Configure the Java Message Service Queue (JMSQ) Listener. The following table lists and describes the parameters that you must configure for the JMSQ Listener.

Parameter	Value	Description
Connection Factory	ConnectionFactory	As defined in the JNDI .bindings file.
JNDI Factory	com.sun.jndi.fscontext.RefFSContextFactory	This value is specified since fscontext is being used.
JNDI URL	file:/c:/working	This is a file URL that points to the directory where the .bindings file was saved. Note that this value does not point directly to the file, but just to the directory.

		Note: This parameter must use a file URL (for example, c:/…). If you do not want to use a file URL, see the JNDI file system documentation (*jndi-fs.html*) that is included with the *fscontext-1_2-beta3.zip* package.
User	guest	This is the default user ID being used unless you changed the RabbitMQ security settings.
Password	guest	This is the default password being used unless you changed the RabbitMQ security settings.
Receiver Name	RequestQ	As defined in the JNDI .bindings file.
Default Reply	ResponseQ	As defined in the JNDI .bindings file.
Default Output Style	Queue	
Output Message Type	Dynamic	

Configuring a Process Flow With the File Emit Service

Create a process flow for the JMSQ Listener that also includes the File Emit Service (com.ibi.agents.XDFileEmitAgent). This will write the messages received by the JMSQ Listener to a directory where you can examine the messages.

The following table lists and describes the parameters that you must configure for the File Emit Service (com.ibi.agents.XDFileEmitAgent).

Parameter	Value	Description
Source of Data	Blank (no value)	Determines the source of data to emit. You can leave this value blank (no value), so the input of the service is written.
Target Directory	c:\working\f1\out	The target output directory. You can provide any directory you choose.
File Pattern	fromRMQ_###.out	The file name pattern.
Return	Status (default)	The status document will be the output document.

With the configuration that is described in this how-to, you can send a message to your File Listener, which will send it to *RequestQ*. The JMSQ Listener will pick up the message from *RequestQ*, write a copy of the message to a file, and then send the message to *ResponseQ*.

As you send more messages, you will notice that the number of messages on *ResponseQ* increases.

Working With Rabbit Advanced Message Queuing Protocol (AMQP) Queues

In RabbitMQ, some common attributes of JMS queues, including message priority, are not supported by default. For example, for priority support, a queue must be created with the *x-max-priority* argument, as described in:

https://www.rabbitmq.com/priority.html

This is also true for queues that specify maximum message length, time to live, and other properties.

With a JNDI binding, as shown in the examples in this how-to, a JMS client cannot connect to a queue that was created with any of those attributes. This happens because of the way an AMQP client declares a destination. The declaration specifies the name of the desired queue, along with any additional attributes required to describe the queue. If no queue with the supplied name exists, then a new queue is created. If a queue with the supplied name does exist, and has the same attributes given in the declaration, then the existing queue is used. However, if the queue exists, but its attributes do not match the declaration, then the broker assumes that the client is trying to modify an existing queue, which is not allowed.

Since a JMS client does not know how to define an AMQP queue, the RabbitMQ JMS connector will declare the destination by name, without any additional attributes. If the existing queue has any such attributes, then the declaration will fail. To avoid this issue, set the *amqp* property of your JMS destination. This informs the RabbitMQ JMS connector that you are trying to connect to an existing RabbitMQ destination and that no declaration is necessary. Along with the *amqp* property, the following settings are required:

- **amqpQueueName**. The name of the queue in RabbitMQ.
- **amqpExchangeName.** The name of the AMQP exchange to which the message should be sent.
- **amqpRoutingKey.** The routing key to send with the message.

To use the default exchange, leave *amqpExchangeName* empty and specify the queue name as *amqpRoutingKey*. The following is an example of the binding for a priority queue in an fscontext .bindings file, using the default exchange:

```
PriorityQ/ClassName=javax.jms.Queue
PriorityQ/FactoryName=com.rabbitmq.jms.admin.RMQObjectFactory
PriorityQ/RefAddr/0/Content=jms/Queue
PriorityQ/RefAddr/0/Type=name
PriorityQ/RefAddr/0/Encoding=String
PriorityQ/RefAddr/1/Content=javax.jms.Queue
PriorityQ/RefAddr/1/Type=type
PriorityQ/RefAddr/1/Encoding=String
PriorityQ/RefAddr/2/Content=com.rabbitmq.jms.admin.RMQObjectFactory
PriorityQ/RefAddr/2/Type=factory
PriorityQ/RefAddr/2/Encoding=String
PriorityQ/RefAddr/3/Content=priorityQ
PriorityQ/RefAddr/3/Type=destinationName
PriorityQ/RefAddr/3/Encoding=String
PriorityQ/RefAddr/4/Content=true
PriorityQ/RefAddr/4/Type=amqp
PriorityQ/RefAddr/4/Encoding=String
PriorityQ/RefAddr/5/Content=priorityQ
PriorityQ/RefAddr/5/Type=amqpQueueName
PriorityQ/RefAddr/5/Encoding=String
PriorityQ/RefAddr/6/Content=
PriorityQ/RefAddr/6/Type=amqpExchangeName
PriorityQ/RefAddr/6/Encoding=String
```

```
PriorityQ/RefAddr/7/Content=priorityQ
PriorityQ/RefAddr/7/Type=amqpRoutingKey
PriorityQ/RefAddr/7/Encoding=String
```

Within iWay Service Manager (iSM), no additional configuration should be required to use these queues, once the binding is configured correctly.