Using the Java Messaging System

Table of Contents

Overview 1
WSDL Namespace 1
Configuration Namespace 1
Basic Endpoint Configuration 1
Using WSDL 2
Using Celtix Configuration 3
Consumer Endpoint Configuration 4
Using Celtix Configuration 5
Using WSDL 6
Service Endpoint Configuration 6
Using Celtix Configuration 6
Using WSDL 7
Using the JMS Context 7
Inspecting JMS Properties 7
Setting JMS Properties 9

Overview

Celtix provides a transport plug-in that enables endpoints to use Java Messaging System (JMS) queues and topics. Celtix's JMS transport plug-in uses the Java Naming and Directory Interface (JNDI) to locate and obtain references to the JMS provider that brokers for the JMS destinations. Once Celtix has established a connection to a JMS provider, Celtix supports the passing of messages packaged as either a JMS `ObjectMessage` or a JMS `TextMessage`.

WSDL Namespace

The WSDL extensions for defining a JMS endpoint are defined in the namespace `http://celtix.objectweb.org/transports/jms`. In order to use the JMS extensions you will need to add the line shown in Text 1 to the `definitions` element of your contract.

```
xmlns:jms="http://celtix.objectweb.org/transports/jms"
```

Text 1: JMS Extension Namespace

Configuration Namespace

The Celtix JMS endpoint configuration properties are specified under the namespace http://celtix.objectweb.org/transports/jms. In order to use the JMS configuration properties you will need to add the line shown in Text 2 to the `beans` element of your configuration.

```
xmlns:jms="http://celtix.objectweb.org/transports/jms"
```

Text 2: JMS Properties Namespace

Basic Endpoint Configuration

JMS endpoints need to know certain basic information about how to establish a connection to the proper destination. This information can be provided in one of two places:

- WSDL File
- Celtix Configuration

Using WSDL

The JMS destination information is provided using the `jms:address` element and its child, the `jms:JMSNamingProperties` element. The `jms:address` element's attributes specify the information needed to identify the JMS broker and the destination. The `jms:JMSNamingProperties` element specifies the Java properties used to connect to the JNDI service.

The address Element

The basic configuration for a JMS endpoint is done by using a `jms:address` element as the child of your service's `port` element. The `jms:address` element uses the attributes described in Table 1 to configure the connection to the JMS broker.

Attribute	Description
`destinationStyle`	Specifies if the JMS destination is a JMS queue or a JMS topic.
`jndiConnectionFactoryName`	Specifies the JNDI name bound to the JMS connection factory to use when connecting to the JMS destination.
`jndiDestinationName`	Specifies the JNDI name bound to the JMS destination to which requests are sent.
`jndiReplyDestinationName`	Specifies the JNDI name bound to the JMS destinations where replies are sent. This attribute allows you to use a user defined destination for replies. For more details see Using a Named Reply Destination.
`connectionUserName`	Specifies the username to use when connecting to a JMS broker.
`connectionPassword`	Specifies the password to use when connecting to a JMS broker.

Table 1: JMS Endpoint Attributes

The JMSNamingProperties Element

To increase interoperability with JMS and JNDI providers, the `jms:address` element has a child element, `jms:JMSNamingProperties`, that allows you to specify the values used to populate the properties used when connecting to the JNDI provider. The `jms:JMSNamingProperties` element has two attributes: `name` and `value`. The `name` attribute specifies the name of the property to set. The `value` attribute specifies the value for the specified property. `jms:JMSNamingProperties` element can also be used for specification of provider specific properties.

The following is a list of common JNDI properties that can be set:

- `java.naming.factory.initial`
- `java.naming.provider.url`
- `java.naming.factory.object`

- java.naming.factory.state
- java.naming.factory.url.pkgs
- java.naming.dns.url
- java.naming.authoritative
- java.naming.batchsize
- java.naming.referral
- java.naming.security.protocol
- java.naming.security.authentication
- java.naming.security.principal
- java.naming.security.credentials
- java.naming.language
- java.naming.applet

For more details on what information to use in these attributes, check your JNDI provider's documentation and consult the Java API reference material.

Using a Named Reply Destination

By default, Celtix endpoints using JMS create a temporary queue for sending replies back and forth. You can change this behavior by setting the `jndiReplyDestinationName` attribute in the endpoint's contract. A Celtix client endpoint will listen for replies on the specified destination and it will specify the value of the attribute in the `ReplyTo` field of all outgoing requests. A Celtix service endpoint will use the value of the `jndiReplyDestinationName` attribute as the location for placing replies if there is no destination specified in the request's `ReplyTo` field.

Example

Text 3 shows an example of an Celtix JMS port specification.

```
<service name="JMSService">
 <port binding="tns:Greeter_SOAPBinding" name="SoapPort">
  <jms:address jndiConnectionFactoryName="ConnectionFactory"
               jndiDestinationName="dynamicQueues/test.Celtix.jmstransport">
   <jms:JMSNamingProperty name="java.naming.factory.initial"
                          value="org.activemq.jndi.ActiveMQInitialContextFactory" />
   <jms:JMSNamingProperty name="java.naming.provider.url"
                          value="tcp://localhost:61616" />
  </jms:address>
 </port>
</service>
```

Text 3: Celtix JMS Port

Using Celtix Configuration

In addition to using the WSDL file to specify the connection information for a JMS endpoint, you can supply it in the endpoint's configuration file. The information in the configuration file will override the information in the endpoint's WSDL file.

JMS connection information is specified using the `jmsAddress` property. The `jmsAddress` property has a single value: `jms:address`. It is identical to the jms:address element used in the WSDL file. Its attributes are listed in Table 1. Like the `jms:address` element in the WSDL file, the `jms:address` property also has a `jms:JMSNamingProperties` child element that is used to specify additional information used to connect to a JNDI provider. For more information on populating this element see JMSNamingProperties.

The `jmsAddress` property can be specified in either the client configuration bean or the service configuration bean.

Example

Text 4 shows a Celtix configuration entry for configuring the addressing information for a JMS consumer endpoint.

```
<beans xmlns:ct="http://celtix.objectweb.org/configuration/types"
       xmlns:jms="http://celtix.objectweb.org/transports/jms">
...
  <bean id="celtix.{http://celtix.objectweb.org/jms_conf_test}
HelloWorldQueueBinMsgService/HelloWorldQueueBinMsgPort.jms-client"
        class="org.objectweb.celtix.bus.transports.jms.jms_client_config.spring.JMSC
lientConfigBean">
    <property name="jmsAddress">
      <value>
        <jms:address destinationStyle="queue"
                     jndiConnectionFactoryName="MockConnectionFactory"
                     jndiDestinationName="myOwnDestination"
                     jndiReplyDestinationName="myOwnReplyDestination"
                     connectionUserName="testUser"
                     connectionPassword="testPassword">
          <jms:JMSNamingProperty name="java.naming.factory.initial"
            value="org.objectweb.celtix.transport.jms.MockInitialContextFactory"/>
          <jms:JMSNamingProperty name="java.naming.provider.url"
                                 value="tcp://localhost:61616"/>
        </jms:address>
      </value>
    </property>
...
</beans>
```

Text 4: Addressing Information in Celtix Configuration

More information

For more information on using Celtix configuration see the Celtix Configuration Guide.

Consumer Endpoint Configuration

JMS consumer endpoints specify the type of messages they use. JMS consumer endpoint can use either a JMS `ObjectMessage` or a JMS `TextMessage`. When using an `ObjectMessage` the consumer endpoint uses a `byte[]` as the method for storing data into and retrieving data from the JMS message body. When messages are sent, the message data, including any formating information, is packaged into a `byte[]` and placed into the JMS

message body before it is placed on the wire. When messages are received, the consumer endpoint will attempt to unmarshall the data stored in the JMS body as if it were packed in a `byte[]`.

When using a `TextMessage`, the consumer endpoint uses a string as the method for storing and retrieving data from the JMS message body. When messages are sent, the message information, including any format-specific information, is converted into a string and placed into the JMS message body. When messages are received the consumer endpoint will attempt to unmashall the data stored in the JMS message body as if it were packed into a string.

When a native JMS applications interact with Celtix consumers, the JMS application is responsible for interpreting the message and the formatting information. For example, if the Celtix contract specifies that the binding used for a JMS endpoint is SOAP, and the messages are packaged as `TextMessage`, the receiving JMS application will get a text message containing all of the SOAP envelope information.

Consumer endpoint can be configured in one of two ways:

- Celtix configuration
- WSDL file

The recommended method is to place the consumer endpoint specific information into the Celtix configuration file for the endpoint.

Using Celtix Configuration

Consumer endpoint configuration is specified using the `org.objectweb.celtix.bus.transports.jms.jms_client_config.spring.JMSClientConfigBean` class for the configuration bean. Using this configuration bean, you specify the message type supported by the consumer endpoint using the `jmsClient` property. It has a single value, `jms:client`, that has a single attribute:

`messageType` Specifies how the message data will be packaged as a JMS message. `text` specifies that the data will be packaged as a `TextMessage`. `binary` specifies that the data will be packaged as an `ObjectMessage`.

Example

Text 5 shows a Celtix configuration entry for configuring a JMS consumer endpoint.

```
<beans xmlns:ct="http://celtix.objectweb.org/configuration/types"
       xmlns:jms="http://celtix.objectweb.org/transports/jms">
...
  <bean id="celtix.{http://celtix.objectweb.org/jms_conf_test}
HelloWorldQueueBinMsgService/HelloWorldQueueBinMsgPort.jms-client"
        class="org.objectweb.celtix.bus.transports.jms.jms_client_config.spring.JMSC
lientConfigBean">
    <property name="jmsClient">
      <value>
        <jms:client messageType="binary" />
      </value>
    </property>
  </bean>
...
</beans>
```

Text 5: Configuration for a JMS consumer endpoint

Adding address information

In addition to specifying the `jmsClient` property, you can also specify the contact information used by the consumer for contacting a service endpoint. This is done by adding the `jmsAddress` property to the consumer endpoint's configuration bean.

More information

For more information on using Celtix configuration see the Celtix Configuration Guide.

Using WSDL

The type of messages accepted by a JMS consumer endpoint is configured using the optional `jms:client` element. The `jms:client` element is a child of the WSDL `port` element and has one attribute:

`messageType`	Specifies how the message data will be packaged as a JMS message. `text` specifies that the data will be packaged as a `TextMessage`. `binary` specifies that the data will be packaged as an `ObjectMessage`.

Service Endpoint Configuration

JMS service endpoints have a number of behaviors that are configurable in the contract. These include:

- how messages are correlated
- the use of durable subscriptions
- if the service uses local JMS transactions
- the message selectors used by the endpoint

Service endpoints can be configure in one of two ways:

- Celtix configuration
- WSDL file

Using Celtix Configuration

Service endpoint configuration is specified using the `org.objectweb.celtix.bus.transports.jms.jms_server_config.spring.JMSServerConfigBean` class for the configuration bean. Using this configuration bean, you specify the service endpoint's behaviors using the `jmsServer` property. It has a single value, `jms:server`, that has a the following attributes:

`useMessageIDAsCorrealationID`	Specifies whether the JMS broker will use the message ID to correlate messages. The default is `false`.
`durableSubscriberName`	Specifies the name used to register a durable subscription.
`messageSelector`	Specifies the string value of a message selector to use. For more information on the syntax used to specify message selectors, see the JMS 1.1 specification.
`transactional`	Specifies whether the local JMS broker will create transactions around message processing. The default is `false`. Currently, this is not supported by the runtime.

Example

Text 6 shows a Celtix configuration entry for configuring a JMS service endpoint.

```
<beans xmlns:ct="http://celtix.objectweb.org/configuration/types"
       xmlns:jms="http://celtix.objectweb.org/transports/jms">
...
  <bean id="celtix.{http://celtix.objectweb.org/jms_conf_test}
HelloWorldQueueBinMsgService/HelloWorldQueueBinMsgPort.jms-server"
        class="org.objectweb.celtix.bus.transports.jms.jms_server_config.spring.JMSS
erverConfigBean">
    <property name="jmsServer">
      <value>
        <jms:server messageSelector="pickMe"
                    useMessageIDAsCorrelationID="true"
                    transactional="false"
                    durableSubscriberName="CeltixSubscriber" />
      </value>
    </property>
  </bean>
...
</beans>
```

Text 6: Configuration for a JMS service endpoint

Adding address information

In addition to specifying the `jmsServer` property, you can also specify the contact information of the service endpoint. This is done by adding the `jmsAddress` property to the service endpoint's configuration bean.

More information

For more information on using Celtix configuration see the Celtix Configuration Guide.

Using WSDL

Service endpoint behaviors are configured using the optional `jms:server` element. The `jms:server` element is a child of the WSDL `port` element and has the following attributes:

`useMessageIDAsCorrealationID`	Specifies whether JMS will use the message ID to correlate messages. The default is `false`.
`durableSubscriberName`	Specifies the name used to register a durable subscription.
`messageSelector`	Specifies the string value of a message selector to use. For more information on the syntax used to specify message selectors, see the JMS 1.1 specification.
`transactional`	Specifies whether the local JMS broker will create transactions around message processing. The default is `false`. Currently, this is not supported by the runtime.

Using the JMS Context

The Celtix context mechanism can be used to inspect a number of the properties associated with a JMS message. The context mechanism can also be used to override some of the JMS endpoint's configuration.

Inspecting JMS Properties

Once a message has been successfully retrieved from the JMS transport you can inspect the JMS header properties using the consumer's response context. In addition, you can see how long the client will wait for a response before timing out.

JMS Header Properties

The JMS properties table lists the properties in the JMS header that you can inspect. The JMS header properties are retrieved using the response context's `get()` method and supplying the value `org.objectweb.celtix.jms.client.response.headers`. The returned properties will be of type `JMSMessageHeadersType`.

Property Name	Property Type	Getter Method
Correlation ID	`string`	`getJMSCorralationID()`
Delivery Mode	`int`	`getJMSDeliveryMode()`
Message Expiration	`long`	`getJMSExpiration()`
Message ID	`string`	`getJMSMessageID()`
Priority	`int`	`getJMSPriority()`
Redelivered	`boolean`	`getJMSRedlivered()`
Time Stamp	`long`	`getJMSTimeStamp()`
Type	`string`	`getJMSType()`
Time To Live	`long`	`getTimeToLive()`

Table 1.: JMS Header Properties

In addition, you can inspect any optional properties stored in the JMS header using `JMSMessageHeadersType.getProperty()`. The optional properties are returned as a `List` of `JMSPropertyType`. Optional properties are stored as name/value pairs.

Example

Text 7 shows code for inspecting some of the JSM properties using the response context.

```
// Proxy greeter initialized previously
InvocationHandler handler = Proxy.getInvocationHandler(greeter);

// Invoke on greeter proxy

BindingProvider bp= null;
if (handler instanceof BindingProvider)
{
  bp = (BindingProvider)handler;
  Map<String, Object> responseContext = bp.getResponseContext();

  JMSMessageHeadersType responseHdr = (JMSMessageHeadersType)
```

```
    responseContext.get("org.objectweb.celtix.jms.client.response.headers");
  if (responseHdr != null)
  {
    System.out.println("Correlation ID: "+responseHdr.getJMSCorrelationID());
    System.out.println("Message Priority: "+responseHdr.getJMSPriority());
    System.out.println("Redelivered: "+responseHdr.getRedelivered());
  }
}
```

Text 7: Reading JMS Properties from the Response Context

The code in Text 7 does the following:

1. Gets the `InvocationHandler` for the proxy whose message headers you want to inspect.
2. Casts the returned `InvocationHandler` object into a `BindingProvider` object to retrieve the response context.
3. Gets the response context.
4. Retrieves the JMS message headers from the response context.
5. Prints some of the message header properties.

Setting JMS Properties

Using the request context in a consumer endpoint, you can set a number of the JMS message header properties and the consumer endpoint's timeout value. These properties are valid for a single invocation. You will need to reset them each time you invoke an operation on the service proxy.

JMS Header Properties

The JMS properties table lists the properties in the JMS header that you can set using the consumer endpoint's request context.

Property Name	Property Type	Setter Method
Correlation ID	`string`	`setJMSCorralationID()`
Delivery Mode	`int`	`setJMSDeliveryMode()`
Time To Live	`long`	`setTimeToLive()`

Table 2.: Settable JMS Header Properties

To set these properties do the following:

1. Create a `JMSMessageHeadersType` object.
2. Populate the values you wish to set using the appropriate setter methods from the JMS properties table.
3. Set the values into the request context by calling the request context's `put()` method using `org.objectweb.celtix.jms.client.request.headers` as the first argument and the new `JMSMessageHeadersType` object as the second argument.

Optional JMS Header Properties

You can also set optional properties into the JMS header. Optional JMS header properties are stored in the `JMSMessageHeadersType` object that is used to set the other JMS header properties. They are stored as a `List` of `JMSPropertyType`. To add optional properties to the JMS header do the following:

1. Create a `JMSPropertyType` object.
2. Set the property's name field using `setName()`.
3. Set the property's value field using `setValue()`.
4. Add the property to the JMS message header to the JMS message header using `JMSMessageHeadersType.getProperty().add(JMSPropertyType)`.
5. Repeat steps 1 through 4 until all of the properties have been added to the message header.

Client Receive Timeout

In addition to the JMS header properties, you can set the amount of time a consumer endpoint will wait for a response before timing out. You set the value by calling the request context's `put()` method with `org.celtix.jms.client.timeout` as the first argument and a `long` representing the amount of time in milliseconds that you want to consumer to wait as the second argument.

Example

Text 8 shows code for setting some of the JMS properties using the request context.

```
// Proxy greeter initialized previously
InvocationHandler handler = Proxy.getInvocationHandler(greeter);

BindingProvider bp= null;
if (handler instanceof BindingProvider)
{
  bp = (BindingProvider)handler;
  Map<String, Object> requestContext = bp.getRequestContext();

  JMSMessageHeadersType requestHdr = new JMSMessageHeadersType();
  requestHdr.setJMSCorrelationID("WithBob");
  requestHdr.setJMSExpiration(3600000L);

  JMSPropertyType prop = new JMSPropertyType;
  prop.setName("MyProperty");
  prop.setValue("Bluebird");
  requestHdr.getProperty().add(prop);

  requestContext.put("org.objectweb.celtix.jms.client.request.headers",
                     requestHdr);

  requestContext.put("org.objectweb.celtix.client.timeout", new Long(1000));
}
```

Text 8: Setting JMS Properties using the Request Context

The code in Text 8 does the following:

1. Gets the `InvocationHandler` for the proxy whose JMS properties you want to change.
2. Casts the returned `InvocationHandler` object into a `BindingProvider` object to retrieve the request context.

3. Gets the request context.
4. Creates a JMSMessageHeadersType object to hold the new message header values.
5. Sets the Correlation ID.
6. Sets the Expiration property to 60 minutes.
7. Creates a new JMSPropertyType object.
8. Sets the values for the optional property.
9. Adds the optional property to the message header.
10. Sets the JMS message header values into the request context.
11. Sets the client receive timeout property to 1 second.