SOAP over JMS 1.0 support

The JMS Transport offers an alternative messaging mechanism to SOAP over HTTP. SOAP over JMS offers more reliable and scalable messaging support than SOAP over HTTP. The SOAP over JMS specification is aimed at a set of standards for the transport of SOAP messages over JMS. Its main purpose is to ensure interoperability between the implementations of different Web services vendors. CXF supports and is compliant with this specification.

SOAP over JMS Namespace

JMS URI

JMS endpoints need to know the address information for establishing connections to the proper destination. SOAP over JMS implements the URI Scheme for Java Message Service 1.0.

JMS URI Scheme

```
jms:<variant>:<destination name>?param1=value1&param2=value2
```

Variants

Prefix	Description
jndi	Destination name is a jndi queue name
jndi-topic	Destination name is a jndi topic name
queue	Destination is a queue name resolved using JMS
topic	Destination is a topic name resolved using JMS

Further parameters can be added as query parameters in the URI.

For example:

```
jms:jndi:SomeJndiNameForDestination?jndiInitialContextFactory=org.apache.activemq.jndi.
ActiveMQInitialContextFactory&jndiURL=tcp://localhost:61616&priority=3
jms:queue:ExampleQueueName?timeToLive=1000
```

JMS parameters

Query Parameter	From Version	DefaultValue	Description
conduitIdSelectorPrefix	3.0.0		If set then this string will be the prefix for all correlation ids the conduit creates and also be used in the selector for listening to replies
deliveryMode		PERSISTENT	NON_PERSISTENT messages will kept only in memory PERSISTENT messages will be saved to disk
durableSubscriptionClientId	3.0.1		Optional Client identifier for the connection. The purpose is to associate a connection with a state maintained on behalf of the client by a provider. The only such state identified by the JMS API is that required to support durable subscriptions.
durableSubscriptionName	3.0.0		
jndiConnectionFactoryName		ConnectionFactory	Specifies the JNDI name bound to the JMS connection factory to use when connecting to the JMS destination.
jndiInitialContextFactory			Specifies the fully qualified Java class name of the "InitialContextFactory" implementation class to use.
jndiTransactionManagerName	3.0.0		Name of the JTA TransactionManager. Will be searched in spring, blueprint and jndi. If a transaction manager is found then JTA transactions will be enabled. See details below.
jndiURL			Specifies the JNDI provider URL
jndi-*			Additional parameters for a JNDI provider
messageType	3.0.0	byte	JMS message type used by CXF (byte, text or binary)
password	3.0.0		Password for creating the connection. Using this in the URI is discouraged

priority	3.0.0	4	Priority for the messages. See your JMS provider documentation for details. Values range from 0 to 9 where 0 is lowest priority
replyToName			Specifies the JNDI name bound to the JMS destinations where replies are sent
receiveTimeout	3.0.0	60000	Timeout in milliseconds the client waits for a reply in case of request / repy exchanges
reconnectOnException	deprecated in 3.0.0	true	Should the transport reconnect in case of exceptions. From version 3.0.0 on the transport will always reconnect in case of exceptions
sessionTransacted	3.0.0	false	Set to true for resource local transactions. Do not set if you use JTA
timeToLive		0	Time (in ms) after which the message will be discarded by the jms provider
topicReplyToName			Reply to messages on a topic with this name. Depending on the variant this is either a jndi or jms name.
useConduitIdSelector	3.0.0	true	Each conduit is assigned with a UUID. If set to true this conduit id will be the prefix for all correlation ids. This allows several endpoints to share a JMS queue or topic
username	3.0.0		Username for creating the connection
concurrentConsumers		1	Number of consumers listening queue concurrently

Some of these attributes are specified in the JMS URI specification.

WSDL Extension

The WSDL extensions for defining a JMS endpoint use a special namespace. In order to use the JMS WSDL extensions you will need to add the namespace definition shown below to the definitions element of your contract.

```
xmlns:soapjms="http://www.w3.org/2010/soapjms/"
```

Various JMS properties may be set in three places in the WSDL — the binding, the service, and the port. Values specified at the service will propagate to all ports. Values specified at the binding will propagate to all ports using that binding.
For example, if the **jndiInitialContextFactory** is indicated for a service, it will be used for all of the port elements it contains.

JMS Properties. For details refer to the URI query parameters with the same name:

Name
deliveryMode
jndiConnectionFactoryName
jndiInitialContextFactory
jndiURL
replyToName
priority
timeToLive
jndiContextParameter

Here is an example:

Ways to define a Service with JMS transport

```
<wsdl11:binding name="exampleBinding">
  <soapjms:jndiContextParameter name="name" value="value" />
  <soapjms:jndiConnectionFactoryName>ConnectionFactory</soapjms:jndiConnectionFactoryName>
  <soapjms:jndiInitialContextFactory>org.apache.activemq.jndi.ActiveMQInitialContextFactory</soapjms:
jndiInitialContextFactory>
  <soapjms:jndiURL>tcp://localhost:61616</soapjms:jndiURL>
  <soapjms:deliveryMode>PERSISTENT</soapjms:deliveryMode>
  <soapjms:priority>5</soapjms:priority>
  <soapjms:timeToLive>200</soapjms:timeToLive>
</wsdl11:binding>

<wsdl11:service name="exampleService">
  <soapjms:jndiInitialContextFactory>com.example.jndi.InitialContextFactory</soapjms:jndiInitialContextFactory>
  <soapjms:timeTolive>100</soapjms:timeToLive>
  <wsdl11:port name="quickPort" binding="tns:exampleBinding">
    <soapjms:timeToLive>10</soapjms:timeToLive>
  </wsdl11:port>
  <wsdl11:port name="slowPort" binding="tns:exampleBinding">
    ...
  </wsdl11:port>
</wsdl11:service>
```

If a property is specified at multiple levels, the setting at the most granular level takes precedence (port first, then service, then binding). In the above example, notice the timeToLive property — for the quickPort port, the value will be 10ms (specified at the port level). For the slowPort port, the value will be 100ms (specified at the service level). In this example, the setting in the binding will always be overridden.

WSDL Usage

For this example:

Greeter Service with JMS transport

```
<wsdl:definitions name="JMSGreeterService" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://cxf.apache.org/jms_greeter" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:x1="http://cxf.apache.org/jms_greeter/types"
  xmlns:soapjms="http://www.w3.org/2010/soapjms/" name="JMSGreeterService"
  targetNamespace="http://cxf.apache.org/jms_greeter">
    ...
        <wsdl:binding name="JMSGreeterPortBinding" type="tns:JMSGreeterPortType">
                <soap:binding style="document" transport="http://www.w3.org/2010/soapjms/" />
                <soapjms:jndiContextParameter name="name" value="value" />
                <soapjms:jndiConnectionFactoryName>ConnectionFactory</soapjms:jndiConnectionFactoryName>
                <soapjms:jndiInitialContextFactory>org.apache.activemq.jndi.ActiveMQInitialContextFactory<
/soapjms:jndiInitialContextFactory>
                <soapjms:jndiURL>tcp://localhost:61616</soapjms:jndiURL>
                <soapjms:deliveryMode>PERSISTENT</soapjms:deliveryMode>
                <soapjms:priority>5</soapjms:priority>
                <soapjms:timeToLive>1000</soapjms:timeToLive>
                <wsdl:operation name="greetMe">
                        <soap:operation soapAction="test" style="document" />
                        <wsdl:input name="greetMeRequest">
                                <soap:body use="literal" />
                        </wsdl:input>
                        <wsdl:output name="greetMeResponse">
                                <soap:body use="literal" />
                        </wsdl:output>
                </wsdl:operation>
        </wsdl:binding>
    <wsdl:service name="JMSGreeterService">
                <soapjms:jndiConnectionFactoryName>ConnectionFactory</soapjms:jndiConnectionFactoryName>
                <soapjms:jndiInitialContextFactory>org.apache.activemq.jndi.ActiveMQInitialContextFactory<
/soapjms:jndiInitialContextFactory>
                <wsdl:port binding="tns:JMSGreeterPortBinding" name="GreeterPort">
                        <soap:address location="jms:jndi:dynamicQueues/test.cxf.jmstransport.queue" />
                </wsdl:port>
        </wsdl:service>
</wsdl:definitions>
```

- The transport URI (http://www.w3.org/2010/soapjms/) is defined in the <soap:binding>.
- The jms: URI is defined in the <soap:address>
- The extension properties are in the <soap:binding>

Define service endpoint or proxy in spring or blueprint

The JAXWS endpoint or proxy can be defined like in the SOAP/HTTP case. Just use a jms: uri like described above.

In CXF 3 it is possible to omit the jndi settings. Just specify an endpoint like this:

Endpoint in spring

```
<bean id="ConnectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">
  <property name="brokerURL" value="tcp://localhost:61616"/>
</bean>
<jaxws:endpoint id="CustomerService"
  address="jms:queue:test.cxf.jmstransport.queue?timeToLive=1000"
  implementor="com.example.customerservice.impl.CustomerServiceImpl">
</jaxws:endpoint>
```

or a Client like this:

Proxy in spring

```
<bean id="ConnectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">
  <property name="brokerURL" value="tcp://localhost:61616"/>
</bean>
<jaxws:client id="CustomerService"
  address="jms:queue:test.cxf.jmstransport.queue?timeToLive=1000"
  serviceClass="com.example.customerservice.CustomerService">
</jaxws:client>
```

The connection factory will be looked up as a bean in the context. By default the name "ConnectionFactory" is assumed but it can be configured using the jndiConnectionFactoryName uri parameter.

Alternatively the connection factory can be set using the ConnectionFactoryFeature.

Publishing a service with the JAVA API

Developers who don't wish to modify the WSDL file can also publish the endpoint information using Java code. For CXF's SOAP over JMS implementation you can write the following:

```
  // You just need to set the address with JMS URI
  String address = "jms:jndi:dynamicQueues/test.cxf.jmstransport.queue3"
      + "?jndiInitialContextFactory"
      + "=org.apache.activemq.jndi.ActiveMQInitialContextFactory"
      + "&jndiConnectionFactoryName=ConnectionFactory&jndiURL=tcp://localhost:61500";
  Hello implementor = new HelloImpl();
  JaxWsServerFactoryBean svrFactory = new JaxWsServerFactoryBean();
  svrFactory.setServiceClass(Hello.class);
  svrFactory.setAddress(address);
  // And specify the transport ID with SOAP over JMS specification
  svrFactory.setTransportId(JMSSpecConstants.SOAP_JMS_SPECIFICATION_TRANSPORTID);
  svrFactory.setServiceBean(implementor);
  svrFactory.create();

  // Alternatively using JAXWS Endpoint.create and avoiding JNDI
  ConnectionFactory cf = new ActiveMQConnectionFactory("tcp://localhost:61500");
  EndpointImpl ep = (EndpointImpl)Endpoint.create(impl);
  ep.getFeatures().add(new ConnectionFactoryFeature(cf));
  ep.publish("jms:queue:test.cxf.jmstransport.queue?timeToLive=1000");
```

NOTE: For tests it can be useful to create an embedded broker like this:

```
    public final void run() {
        try {
            broker = new BrokerService();
            broker.setPersistent(false);
            broker.setPersistenceAdapter(new MemoryPersistenceAdapter());
            broker.setTmpDataDirectory(new File("./target"));
            broker.setUseJmx(false);
            if (brokerName != null) {
                broker.setBrokerName(brokerName);
            }
            broker.addConnector(brokerUrl1);
            broker.start();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

Consume the service with the API

Sample code to consume a SOAP-over-JMS service is as follows:

```
    public void invoke() throws Exception {
        // You just need to set the address with JMS URI
        String address = "jms:jndi:dynamicQueues/test.cxf.jmstransport.queue3"
            + "?jndiInitialContextFactory=org.apache.activemq.jndi.ActiveMQInitialContextFactory"
            + "&jndiConnectionFactoryName=ConnectionFactory"
            + "&jndiURL=tcp://localhost:61500";
        JaxWsProxyFactoryBean factory = new JaxWsProxyFactoryBean();
        // And specify the transport ID with SOAP over JMS specification
        factory.setTransportId(JMSSpecConstants.SOAP_JMS_SPECIFICIATION_TRANSPORTID);
        factory.setServiceClass(Hello.class);
        factory.setAddress(address);
        Hello client = (Hello)factory.create();
        String reply = client.sayHi(" HI");
        System.out.println(reply);
    }

  // Alternatively using the JAXWS API with jms details defined in WSDL while avoiding JNDI
  SOAPService2 service = new SOAPService2(wsdl, serviceName); // Using the generated service
  ConnectionFactory cf = new ActiveMQConnectionFactory("tcp://localhost:61500");
  ConnectionFactoryFeature cff = new ConnectionFactoryFeature(cf);
  Greeter greeter = service.getPort(portName, Greeter.class, cff); // Connection Factory can be set as a
feature in CXF >= 3.0.0
```

If you specify queue or topic as variant and use cxf >= 3.0.0 then the jndi settings are not necessary.

```
svrFactory.setAddress("jms:queue:test.cxf.jmstransport.queue?timeToLive=1000");
// For CXF >= 3.0.0
svrFactory.setFeatures(Collections.singletonList(new ConnectionFactoryFeature(cf)));
```

In this case case the connection factory is supplied using a feature. For CXF 2.x the connection factory can only be supplied using jndi.