PEtALS-BC-SOAP

This document explains how to install, configure and use the petals-bc-soap JBI component.

PEtALS Team

Christophe HAMERLING <christophe.hamerling@ebmwebsourcing.com>

- June 2008 -

Table of Contents

List of Figures

List of Tables

PETALS-BC-SOAP

This binding component allows to interact with external Web Services and to expose JBI services as Web Services.

A JBI `MessageExchange` sent to a ServiceEndpoint (mapped to a Web Service) is transformed into a SOAP message and sent to the linked external web service. A SOAP message received on an exposed web service is transformed into a JBI MessageExchange and sent to the corresponding JBI ServiceEndpoint.

If you want more details about SOAP, you can consult this W3C specification : http://www.w3.org/TR/soap/

Chapter 1. Features

The petals-bc-soap is based on the petals-cdk v4.0, Apache Axis2 v1.3 and Mortbay Jetty v6.1.4. It provides the following features :

- Expose JBI Services as Web Services
- Expose JBI Services as REST Services
- Expose Web Services as JBI Services
- Expose REST Services as JBI Services
- Handle SOAP attachments. The attachments of the incoming SOAP message are placed into the JBI message as attachments; the JBI attachments are placed in the outgoing SOAP message as attachments.
- WS-notification. The component can send/receive web service notifications to/from external subscribers/producers.
- WS-Security and WS-SecureConversation via the addition of the Rampart's Axis2 module.

Chapter 2. Component Configuration

The component can be configured through its JBI descriptor file like this :

```
<?xml version="1.0" encoding="UTF-8"?>
<jbi:jbi version="1.0" xmlns:jbi="http://java.sun.com/xml/ns/jbi"
 xmlns:petalsCDK="http://petals.ow2.org/components/extensions/version-4.0"
 xmlns:soap="http://petals.ow2.org/components/soap/version-3.1">
 <jbi:component type="binding-component"
  bootstrap-class-loader-delegation="parent-first">
  <jbi:identification>
   <jbi:name>petals-bc-soap</jbi:name>
   <jbi:description> The SOAP Binding Component (based on Axis2 + Jetty)</jbi:description>
  </jbi:identification>
  <jbi:component-class-name>org.ow2.petals.binding.soap.SoapComponent</jbi:component-class-name>
  <jbi:component-class-path>...</jbi:component-class-path>
  <jbi:bootstrap-class-name>org.ow2.petals.binding.soap.SoapBootstrap</jbi:bootstrap-class-name>
  <jbi:bootstrap-class-path>...</jbi:bootstrap-class-path>

  <!-- Component Development Kit Parameters -->
  <petalsCDK:acceptor-pool-size>5</petalsCDK:acceptor-pool-size>
  <petalsCDK:processor-pool-size>10</petalsCDK:processor-pool-size>
  <petalsCDK:ignored-status>DONE_AND_ERROR_IGNORED</petalsCDK:ignored-status>
  <petalsCDK:properties-file />
  <petalsCDK:performance-notifications>false</petalsCDK:performance-notifications>

 <petalsCDK:jbi-listener-class-name>org.ow2.petals.binding.soap.listener.outgoing.JBIListener</
petalsCDK:jbi-listener-class-name>

 <petalsCDK:external-listener-class-
name>org.ow2.petals.binding.soap.listener.incoming.SoapExternalListener</petalsCDK:external-
listener-class-name>

  <!-- SOAP Component Parameters -->
  <soap:http-port>8084</soap:http-port>
  <soap:http-services-list>true</soap:http-services-list>
  <soap:http-services-context>petals</soap:http-services-context>
  <soap:http-services-mapping>services</soap:http-services-mapping>
  <soap:http-thread-pool-size-min>2</soap:http-thread-pool-size-min>
  <soap:http-thread-pool-size-max>50</soap:http-thread-pool-size-max>
  <soap:http-acceptors>4</soap:http-acceptors>
 </jbi:component>
</jbi:jbi>
```

Warning

The class name values in italic should not be modified by the user.

Table 2.1. Configuration of the component (CDK)

Parameter	Description	Default	Required	Scope
acceptor-pool-size	The size of the thread pool used to accept Message Exchange from the NMR. Once a message is accepted, its processing is delegated to the processor pool thread.	5	Yes	Runtime
processor-pool-size	The size of the thread pool used to process Message Exchanges. Once a message is accepted, its processing is delegated to one of the thread of this pool.	10	Yes	Runtime
performance-notifications	Enable the performance notifications in the component. The CDK proposes to a performance notification feature to the component implementor. If you enable this feature, you must use the related method accessible in the `AbstractComponent` class.	-	No	Runtime
performance-step	When the performance notification feature is enabled, it is possible to define a step on the notifications. When there is an heavy message traffic, it is recommanded to increase this step to avoid performance disturbance.	-	No	Runtime
properties-file	Name of the file containing properties used as reference by other parameters. Parameters reference the property name in the following pattern `${myPropertyName}`. At runtime, the expression is replaced by the value of the property. The value of this parameter is : • an URL • a file relative to the PEtALS installation path	-	No	Installation
ignored-status	When the component receives an acknowledgement message exchange, it can skip the processing of these message according to the type of the acknowledgment. If you decide to not ignore some acknowledgement, the component listeners must take care of them. Accepted values : `DONE_AND_ERROR_IGNORED`, `DONE_IGNORED`, `ERROR_IGNORED` or `NOTHING_IGNORED`	`DONE_AND_ERROR_IGNORED`	Yes	Component
jbi-listener-class-name	Qualified name of the class extending **AbstractJBIListener**	-	Yes	Component
external-listener-class-name	Qualified name of the class extending **AbstractExternalListener**	-	No	Component

Definition of CDK parameter scope :

- *Component* : The parameter has been defined during the development of the component. A user of the component can not change its value.
- *Installation*: The parameter can be set during the installation of the component, by using the installation MBean (see JBI specifications for details about the installation sequence). If the parameter is optional and has not been defined during the development of the component, it is not available at installation time.
- *Runtime* : The paramater can be set during the installation of the component and during runtime. The runtime confguration can be changed using the CDK custom MBean named `RuntimeConfiguration`. If the parameter is optional and has not been defined during the development of the component, it is not available at installation and runtime times.

Table 2.2. Configuration of the component (SOAP)

Parameter	Description	Default	Required
http-port	The port used by the Jetty HTTP server to handle incoming http requests	8084	No
http-services-list	Display the list of exposed services on http://<HOST>:<PORT>/<CONTEXT>/<MAPPING>/listServices	true	No
http-thread-pool-size-min	Minimun size of the Jetty HTTP server thread pool	2	No
http-thread-pool-size-max	Maximun size of the Jetty HTTP server thread pool	50	No
http-acceptors	Number of Jetty HTTP acceptors	4	No
http-services-context	Context of the exposed services	petals	No
http-services-mapping	Mapping of the exposed services	services	No

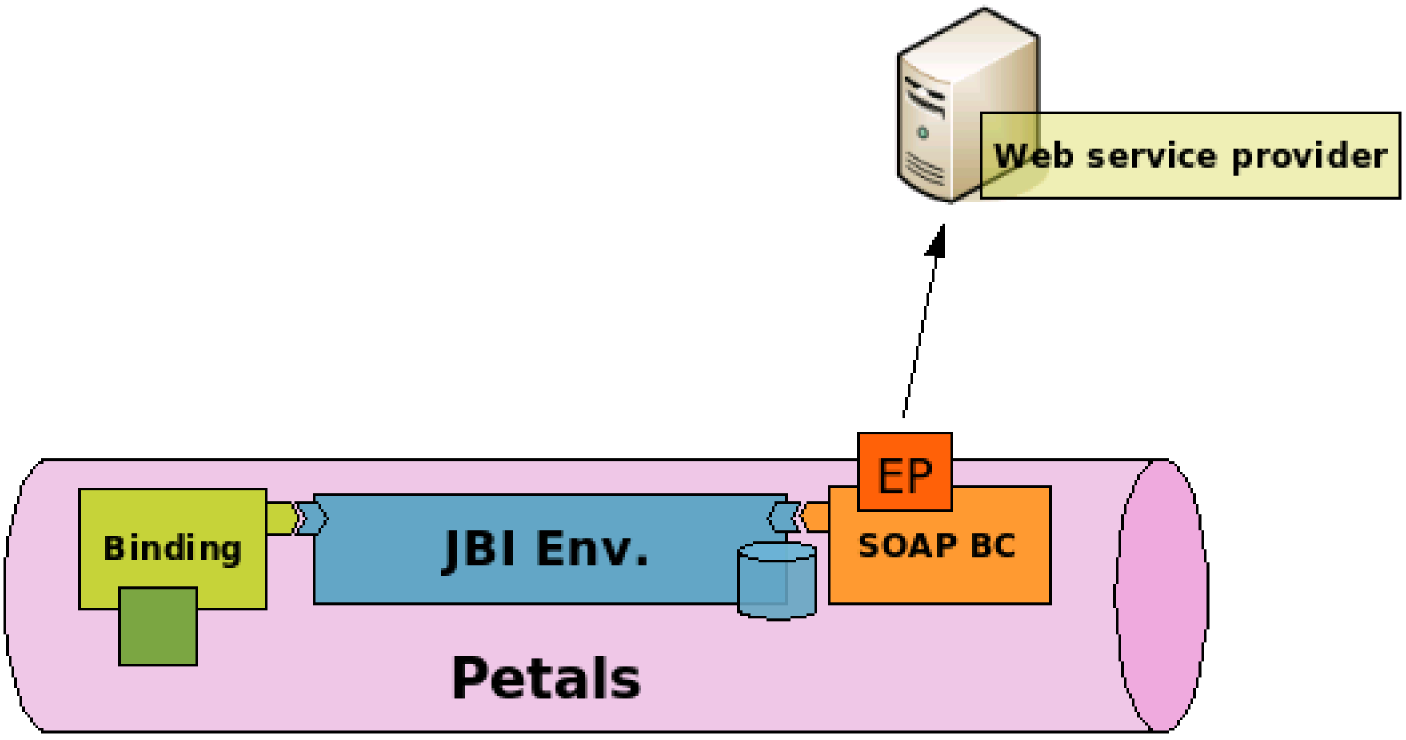
The SOAP component specific parameters can be also set through JMX during its installation phase.
More information about Jetty tunning can be found here.

Chapter 3. Service Configuration

3.1. Send a JBI message to an external Web Service

PROVIDE SERVICE : Expose an external Web Service in the JBI environment

Figure 3.1. Provides an external Web Service as a JBI service



The petals-bc-soap component can expose an external Web Service as a JBI ServiceEndpoint. This is done by deploying a Service Unit on it (see Figure 3.1, “Provides an external Web Service as a JBI service”).

When a message is received on a SOAP linked endpoint from the JBI environment, it is transformed into a SOAP message and sent to the Web Service. The address of the Web Service to send the SOAP message to is defined in the address extension of the deployed Service Unit.

The SOAP message is created like this :

- The JBI message payload is wrapped in the SOAP body
- The JBI message attachments are used to create SOAP ones
- The JBI message exchange operation is used to create the SOAP action
- The JBI MEP is used to determine the SOAP MEP

The external Web Service is called and the SOAP response is processed and returned to the JBI environment.

3.1.1. Service Unit descriptor

The Service Unit descriptor file (`jbi.xml`) looks like this :

```
<?xml version="1.0" encoding="UTF-8"?>
<jbi:jbi version="1.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:jbi="http://java.sun.com/xml/ns/jbi"
 xmlns:petalsCDK="http://petals.ow2.org/components/extensions/version-4.0"
 xmlns:soap="http://petals.ow2.org/components/soap/version-3.1"
 xmlns:sample="http://petals.ow2.org/soap/sample">

 <!-- Import a Service into PEtALS or Expose a PEtALS Service => use a BC. -->
 <jbi:services binding-component="true">

  <!-- Import a Service into PEtALS => provides a Service. -->
  <jbi:provides
   interface-name="sample:SoapInterface"
   service-name="sample:SoapInterface"
   endpoint-name="SoapInterfaceEndpoint">

   <!-- CDK specific fields -->
   <petalsCDK:wsdl>http://example.org/service/SampleWebService?wsdl</petalsCDK:wsdl>

   <!-- SOAP specific fields -->
   <soap:address>http://example.org/service/SampleWebService</soap:address>
   <soap:soap-version>1.1</soap:soap-version>
   <soap:add-root>false</soap:add-root>
   <soap:chunked-mode>false</soap:chunked-mode>
   <soap:cleanup-transport>true</soap:cleanup-transport>
   <soap:mode>SOAP</soap:mode>
  </jbi:provides>
 </jbi:services>
</jbi:jbi>
```

Table 3.1. Configuration of a Service Unit to provide a service (JBI)

Parameter	Description	Default	Required
provides	Describe the JBI service that will be exposed into the JBI bus. `Interface` (qname), `service` (qname) and `endpoint` (string) attributes are required.	-	Yes

Table 3.2. Configuration of a Service Unit to provide a service (CDK)

Parameter	Description	Default	Required
wsdl	Path to the WSDL document describing services and operations exposed by the provided JBI endpoints defined in the SU. The value of this parameter is : • an URL • a file relative to the PEtALS installation path If no wsdl path is specified, a basic description isl automaticaly provided by the CDK.	-	No
timeout	Timeout in milliseconds of a synchronous send. this parameter can be used in conjunction with the `sendSync(Exchange exchange)` method of the Listeners. Set 0 for an infinite timeout.	-	No
org.ow2.petals.messaging.provider.noack	Check PEtALS container document for further details. This propety activates the bypass of acknowledgment messages destinated to this SU.	-	No

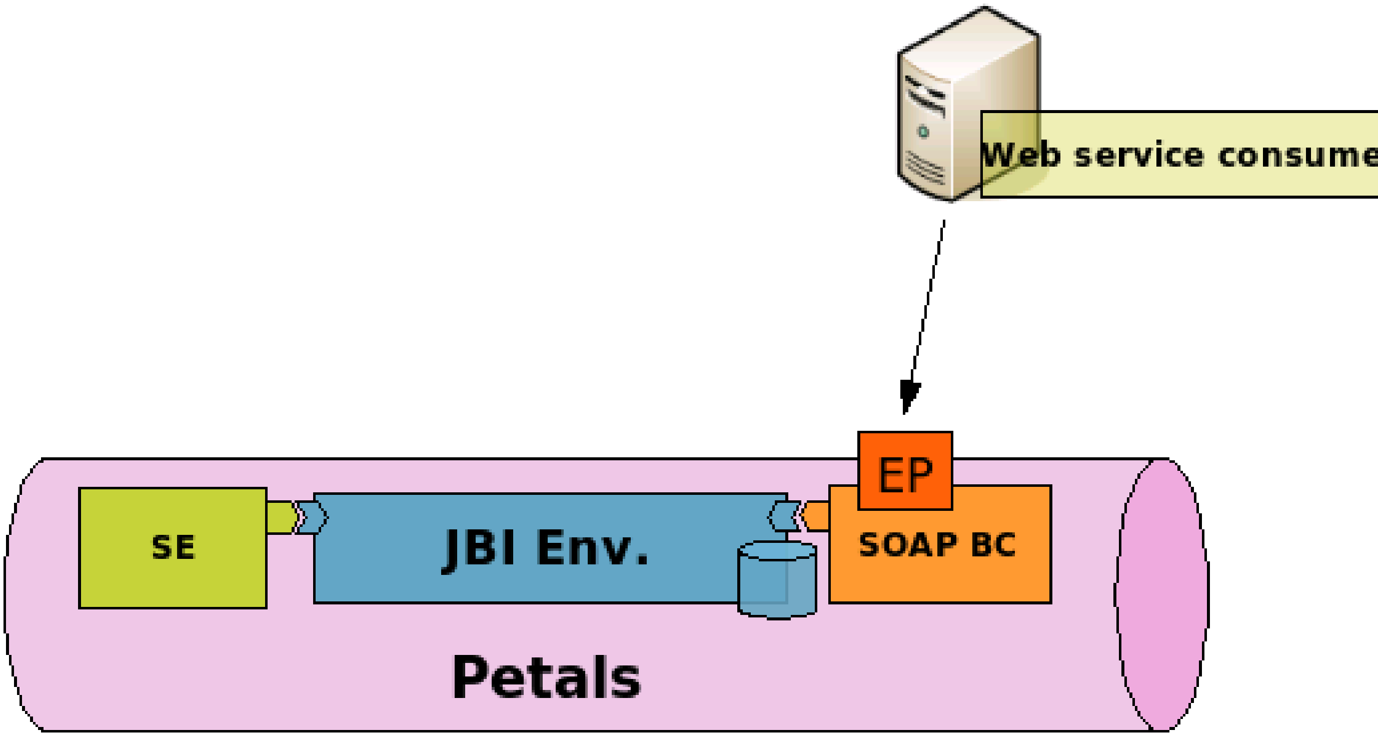
Table 3.3. Configuration of a Service Unit to provide a service (SOAP)

Parameter	Description	Default	Required
address	Address of the external Web Service to send JBI messages to.	-	Yes
mode	The mode to be used to send SOAP message to the specified address. Possible values are : `SOAP` for basic WebService calls, `TOPIC` for WebService notifications and `REST` for REST service calls.	`SOAP`	No
http-method	The HTTP method to be used to send messages to a REST service. Possible values are : `GET`, `POST`, `PUT` and `DELETE`.	`GET`	No
soap-version	The SOAP version used to create SOAP messages. Possible values are `11` and `12`.	`11`	No
timeout	The timeout value. Client invocations will reach timeout after waiting this laps time. The value is expressed in milliseconds.	2000	No
proxy-host	The proxy host name. If it is not set, the proxy mode will be disabled and all others proxy parameters are ignored.	-	No
proxy-port	The proxy host port.	-	No
proxy-user	The proxy user.	-	No
proxy-password	The proxy password.	-	No
proxy-domain	The proxy domain.	-	No
cleanup-transport	Cleanup the transport after the WebService call. Not cleaning up the transport can cause timeouts on large number of calls. Possible values are: `true`, `false`.	`true`	No

3.2. Send a JBI message from an incoming SOAP message

CONSUME SERVICE : Expose an internal service outside of the JBI environment

Figure 3.2. Consumes a JBI service on SOAP message



The petals-bc-soap component can listen incoming SOAP messages and send messages to a JBI ServiceEndpoint. We say that the component consumes the JBI service (see Figure 3.2, “Consumes a JBI service on SOAP message”).

To expose a JBI service as Web Service, you need to deploy a service unit. The address extension value will be used as Axis2 Service name.

When a SOAP message is handled by the Axis2 Service, it is transformed into a JBI Message and sent to the JBI ServiceEndpoint configured in the Service Unit. The JBI message is created like this :

- The JBI operation is created from the SOAP action.
- Copy the SOAP body into the JBI one.
- Put the SOAP attachments into JBI ones.
- Put the SOAP header into the "SOAP.HEADER" JBI message property

The component is configured to handle URIs with the *http://HOST:PORT/petals/services/ADDRESS* pattern. It also handles ?wsdl calls; the wsdl description is retrieved from the endpoint and sent back to the consumer.

Caution

If the service does not provide a WSDL file; the component switch to a dirty mode and always considers that the requested service implements the requested operation. Then, It's the "JBI" container or the service itself which is in charge of verifying that this operation if actualy available.

The list of services is available at *http://HOST:PORT/petals/services/listServices* URI.

3.2.1. Service Unit descriptor

The Service Unit descriptor file (`jbi.xml`) looks like this :

```
<?xml version="1.0" encoding="UTF-8"?>
<jbi:jbi version="1.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:jbi="http://java.sun.com/xml/ns/jbi"
 xmlns:soap="http://petals.ow2.org/components/soap/version-3.1"
 xmlns:petalsCDK="http://petals.ow2.org/components/extensions/version-4.0"
 xmlns:sample="http://petals.ow2.org/soap/sample">

 <!-- Import a Service into PEtALS or Expose a PEtALS Service => use a BC. -->
 <jbi:services binding-component="true">

  <!-- Expose a PEtALS Service => consumes a Service. -->
  <jbi:consumes
   interface-name="sample:ConsumedInterface"
   service-name="sample:ConsumedService"
   endpoint-name="ConsumedEndpoint">

   <!-- CDK specific fields -->
   <petalsCDK:operation>operation</petalsCDK:operation>
   <petalsCDK:mep>InOut</petalsCDK:mep>

   <!-- SOAP specific fields -->
   <soap:address>ExposedService</soap:address>
   <soap:remove-root>false</soap:remove-root>
   <soap:mode>SOAP</soap:mode>
   <!-- optional modules -->
   <soap:modules>addressing, rampart</soap:modules>
  </jbi:consumes>
 </jbi:services>
</jbi:jbi>
```

Table 3.4. Configuration of a Service Unit to consume a service (JBI)

Parameter	Description	Default	Required
consumes	Name of the JBI service to invoke into the JBI bus. You can define only the interface (qname) to let the NMR choose a matching service, or the pair service(qname) and endpoint (string) to consume the localized service.	-	Yes

Table 3.5. Configuration of a Service Unit to consume a service (CDK)

Parameter	Description	Default	Required
mep	Message exchange pattern abbreviation. This parameter can be user in conjunction with the method of the CDK Listeners : `createMessageExchange(Extensions extensions)`. This method returns a CDK Exchange corresponding to the type of the specified pattern. Admitted values are : `InOnly`, `RobustInOnly`, `InOptionalOut` et `InOut`	-	No
operation	Operation to call on a service. This parameter can be used in conjunction with the sending methods of the Listeners. If no operation is specified in the Message Exchange to send, this parameter will be used.	-	No
timeout	Timeout in milliseconds of a synchronous send. this parameter can be used in conjunction with the `sendSync(Exchange exchange)` method of the Listeners. Set 0 for an infinite timeout.	-	No
org.ow2.petals.messaging.consumer.noack	Check PEtALS container document for further details. This propety activates the bypass of acknowledgment messages destinated to this SU.	-	No
org.ow2.petals.routing.strategy	**To be used only in platform (distributed) PEtALS distribution**.Check PEtALS platform documentation for further details. Override the default routing strategy for Message Exchanges sent by this SU	-	No
org.ow2.petals.transport.compress	**To be used only in platform (distributed) PEtALS distribution**.Check PEtALS platform documentation for further details. This property activates the compression of the messages payload when set to `true`.	-	No
org.ow2.petals.transport.qos	**To be used only in platform (distributed) PEtALS distribution**. Check PEtALS platform documentation for further details. This property overrides the default policy of the Quality of Service supported by PEtALS Transporter for Message Exchange sent by this SU.	-	No

Table 3.6. Configuration of a Service Unit to consume a service (SOAP)

Parameter	Description	Default	Required
address	The name of the Axis2 Web Service that will be created on Service Unit deployment. This service is created and linked to the JBI context. Each SOAP/REST message received on this service will be forwarded to the JBI endpoint specified in the consumes element. The WebService will be accessible at http://<HOST>:<POST>/<CONTEXT>/<MAPPING>/<address>.	-	Yes
modules	A list of Axis2 modules names (separated by comas) to be engaged on the new Web Service. These modules must be available in the component context. See managed bootstrap section for more details on how to add a module which is not natively provided by the component.	-	No
service-parameters	Additional XML configuration section as CDATA for created Axis2 service.	-	No
remove-root	Remove the root element of the payload. Possible values are: `true`, `false`.	false	No
mode	The mode to be used to send SOAP message to the specified address. Possible values are : `SOAP` for basic WebService calls, `TOPIC` for WebService notifications and `REST` for REST service calls.	`SOAP`	No

Chapter 4. REST Services

4.1. Introduction

The SOAP binding component provides REST (REpresentational State Transfer (http://en.wikipedia.org/wiki/Representational_State_Transfer) services features since release 3.1. The REST feature is provided by Axis2 in the component.

4.2. Configuration

The component can be configured to :

- Provide access to an external REST Service. This service will be available as JBI service inside the JBI environment. Each JBI message received on the JBI endpoint will be used to invoke the external REST Service. This mode is configured with a Service Unit in "provider mode".
- Expose a JBI Service as REST Service. The JBI Service can be accessed from outside of the JBI environment like other standard REST Services. This mode is configured with a Service Unit in "consumer mode".

4.2.1. Provide mode : Provide access to external REST Service

In order to activate REST mode, the Service Unit (in provide mode) must be configured like

```
<?xml version="1.0" encoding="UTF-8"?>
<jbi:jbi version="1.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:jbi="http://java.sun.com/xml/ns/jbi"
 xmlns:petalsCDK="http://petals.ow2.org/components/extensions/version-4.0"
 xmlns:soap="http://petals.ow2.org/components/soap/version-3.1"
 xmlns:sample="http://petals.ow2.org/soap/sample">

 <!-- Import a Service into PEtALS or Expose a PEtALS Service => use a BC. -->
 <jbi:services binding-component="true">

  <!-- Import a Service into PEtALS => provides a Service. -->
  <jbi:provides
   interface-name="sample:SoapInterface"
   service-name="sample:SoapInterface"
   endpoint-name="SoapInterfaceEndpoint">

   <!-- CDK specific fields -->
   <petalsCDK:mep xsi:nil="true"/>

   <!-- WSDL file -->
   <petalsCDK:wsdl>http://example.org/service/SampleWebService?wsdl</petalsCDK:wsdl>

   <!-- SOAP specific fields -->

 <soap:address>http://example.org/param1={xpathexpression1}&amp;param2={xpathexpression1}</
soap:address>
   <soap:mode>REST</soap:mode>
   <soap:rest-http-method>GET</soap:rest-http-method>
  </jbi:provides>
 </jbi:services>
</jbi:jbi>
```

- The address parameter can be configured with placeholders. In the previous code snippet, the placeholder is `the bracket {}`. The placeholder will be replaced by the result of the XPath expression defined inside of the placeholder. The XPath expression is performed on the content of the incoming JBI message. The placeholders will be replaced in the adress parameter to build the final URI according to the result of the XPath expression.

- The mode parameter must be set to REST to enable REST feature in component.
- Possible http-method parameter values are GET, POST, PUT, DELETE (default is GET). It will be used in provider mode by Axis2 as HTTP method invokation.
  - GET : The JBI message is only used to create the URI of the REST service to be invoked with the placeholders mechanism.
  - POST : The JBI message is sent to the REST service.
  - PUT : The JBI message is sent to the REST service.

4.2.2. Consume mode : Expose JBI Service as as REST Service

```
<jbi:jbi version="1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:jbi="http://java.sun.com/xml/ns/jbi"
 xmlns:petalsCDK="http://petals.ow2.org/components/extensions/version-4.0"
 xmlns:helloworld="http://petals.ow2.org/helloworld"
 xmlns:soap="http://petals.ow2.org/components/soap/version-3.1">

 <!-- Import a Service into PEtALS or Expose a PEtALS Service => use a BC. -->
 <jbi:services binding-component="true">
  <!-- Import a Service into PEtALS => provides a Service. -->
  <jbi:consumes interface-name="helloworld:Helloworld" service-name="helloworld:HelloworldService"
 endpoint-name="HelloworldEndpoint">
   <!-- CDK specific fields -->

   <petalsCDK:mep>InOut</petalsCDK:mep>
   <petalsCDK:operation>getXXX</petalsCDK:operation>
   <!-- SOAP specific fields -->
   <soap:address>RESTServiceName</soap:address>
   <soap:mode>REST</soap:mode>
   <soap:rest-add-namespace-uri>http://petals.ow2.org/soapbc</soap:rest-add-namespace-uri>
   <soap:rest-add-namespace-prefix>ns1</soap:rest-add-namespace-prefix>
   <soap:rest-remove-prefix-on-response>*</soap:rest-remove-prefix-on-response>
  </jbi:consumes>
 </jbi:services>
</jbi:jbi>
```

- The `address` parameter is used to create the Axis2 WebService that will be accessible from outside of the JBI environment. This is the same mechanism as for the standard WebService created without REST mode.
- The `mode` parameter value set to REST enable REST feature on the newly created WebService. Without this parameter, the REST mode is unactive.
- The `rest-add-namespace-uri` parameter is used to add a namespace to the generated JBI message.
- The `rest-add-namespace-prefix` parameter is used to specify the prefix to be used for the namespace specified by the rest.add-namespace-uri parameter. Default value is petalsbcsoaprest.
- The `rest-remove-prefix-on-response` is used to specify the prefix namespaces to be removed on message response. The values have to be specified in Coma Separated Value format like 'ns1,ns2'. The special value '*' is used to remove all the namespaces.

The component will create a JBI message depending on the http-method used in the incoming request :

- GET : A JBI message is created from the URL parameters
- POST/PUT/DELETE : The incoming XML message is used to create the JBI message.

In all the cases the namespaces are added to the JBI message if they are specified in the Service Unit configuration.

The JBI operation is created from the incoming REST query. The operation is extracted from the URL. A URL like `http://<host>:<port>/petals/services/RESTService/operation?param1=value1&param2=value2` will produce the '`operation`' JBI operation.

4.3. Samples

4.3.1. Provide mode

In this sample, we are going to provide the Yahoo Weather Service (http://developer.yahoo.com/weather/) as JBI Service inside the JBI environment. It is possible by configuring a Service Unit in provider mode :

```
<?xml version="1.0" encoding="UTF-8"?>
<jbi:jbi version="1.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:jbi="http://java.sun.com/xml/ns/jbi
 xmlns:petalsCDK="http://petals.ow2.org/components/extensions/version-4.0"
 xmlns:soap="http://petals.ow2.org/components/soap/version-3.1"
 xmlns:sample="http://petals.ow2.org/soap/sample">

 <!-- Import a Service into PEtALS or Expose a PEtALS Service => use a BC. -->
 <jbi:services binding-component="true">

  <!-- Import a Service into PEtALS => provides a Service. -->
  <jbi:provides
   interface-name="sample:YahooWeatherInterface"
   service-name="sample:YahooWeatherService"
   endpoint-name="YahooWeatherEndpoint">

   <!-- CDK specific fields -->
   <petalsCDK:mep xsi:nil="true"/>

   <!-- WSDL file -->
   <petalsCDK:wsdl>Weather.wsdl</petalsCDK:wsdl>

   <!-- SOAP specific fields -->

 <soap:address>http://weather.yahooapis.com/forecastrss?p={/*[local-name()='getWeather'][1]/*[local-
name()='citycode'][1]}&amp;u={/*[local-name()='getWeather'][1]/*[local-name()='unit'][1]}</
soap:address>
   <soap:mode>REST</soap:mode>
   <soap:rest-http-method>GET</soap:rest-http-method>
  </jbi:provides>
 </jbi:services>
</jbi:jbi>
```

When receiving a JBI message on the activated JBI endpoint, the final address will be built from the JBI message payload. For example if the following JBI message :

```
<weat:getWeather xmlns:weat="http://petals.ow2.org/services/weather">
  <citycode>FRXX0099</citycode>
  <unit>c</unit>
</weat:getWeather>
```

is associated with the address parameter value `http://weather.yahooapis.com/forecastrss?p={/*[local-name()='getWeather'][1]/*[local-name()='citycode'][1]}&amp;u={/*[local-name()='getWeather'][1]/*[local-name()='unit'][1]}` will produce the URI `http://weather.yahooapis.com/forecastrss?p=FRXX0099&u=c`.

The JBI message response returned by the Yahoo Weather REST service is :

```
<rss version="2.0" xmlns:yweather="http://xml.weather.yahoo.com/ns/rss/1.0"
 xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#">
  <channel>
    <title>Yahoo! Weather - Toulouse, FR</title>

 <link>http://us.rd.yahoo.com/dailynews/rss/weather/Toulouse__FR/*http://weather.yahoo.com/forecast/
FRXX0099_c.html</link>
```

```
    <description>Yahoo! Weather for Toulouse, FR</description>
    <language>en-us</language>
    <lastBuildDate>Thu, 06 Mar 2008 3:00 pm CET</lastBuildDate>
    <ttl>60</ttl>
    <yweather:location city="Toulouse" country="FR" region=""/>
    <yweather:units distance="km" pressure="mb" speed="kph" temperature="C"/>
    <yweather:wind chill="3" direction="310" speed="37"/>
    <yweather:atmosphere humidity="37" pressure="0" rising="0" visibility="999"/>
    <yweather:astronomy sunrise="7:22 am" sunset="6:50 pm"/>
    <image>
      <title>Yahoo! Weather</title>
      <width>142</width>
      <height>18</height>
      <link>http://weather.yahoo.com/</link>
      <url>http://l.yimg.com/us.yimg.com/i/us/nws/th/main_142b.gif</url>
    </image>
    <item>
      <title>Conditions for Toulouse, FR at 3:00 pm CET</title>
      <geo:lat>43.61</geo:lat>
      <geo:long>1.45</geo:long>

 <link>http://us.rd.yahoo.com/dailynews/rss/weather/Toulouse__FR/*http://weather.yahoo.com/forecast/
FRXX0099_c.html</link>
      <pubDate>Thu, 06 Mar 2008 3:00 pm CET</pubDate>
      <yweather:condition code="28" date="Thu, 06 Mar 2008 3:00 pm CET" temp="8" text="Mostly
 Cloudy"/>
      <description>
        <![CDATA[<img src="http://l.yimg.com/us.yimg.com/i/us/we/52/28.gif" /><br />
        <b>Current Conditions:</b><br />
        Mostly Cloudy, 8 C<BR /><BR />
        <b>Forecast:</b><BR />
        Thu - Mostly Cloudy. High: 10 Low: 4<br />
        Fri - Cloudy. High: 10 Low: 4<br />
        <br />
        <a
 href="http://us.rd.yahoo.com/dailynews/rss/weather/Toulouse__FR/*http://weather.yahoo.com/forecast/
FRXX0099_c.html">Full Forecast at Yahoo! Weather</a><BR/>
        (provided by The Weather Channel)<br/>]]>
      </description>
      <yweather:forecast code="27" date="06 Mar 2008" day="Thu" high="10" low="4" text="Mostly
 Cloudy"/>
      <yweather:forecast code="26" date="07 Mar 2008" day="Fri" high="10" low="4" text="Cloudy"/>
      <guid isPermaLink="false">FRXX0099_2008_03_06_15_0_CET</guid>
    </item>
  </channel>
</rss>
```

4.3.2. Consume mode

In this sample, we are going to expose a JBI service as REST service. The Service Unit configuration which will be used is :

```
<jbi:jbi version="1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:jbi="http://java.sun.com/xml/ns/jbi"
 xmlns:petalsCDK="http://petals.ow2.org/components/extensions/version-4.0"
 xmlns:sample="http://petals.ow2.org/sample"
 xmlns:soap="http://petals.ow2.org/components/soap/version-3.1">

 <!-- Import a Service into PEtALS or Expose a PEtALS Service => use a BC. -->
 <jbi:services binding-component="true">
  <!-- Import a Service into PEtALS => provides a Service. -->
  <jbi:consumes interface-name="sample:SampleProvider"
   service-name="sample:SampleProviderService"
   endpoint-name="SampleProvider">

   <!-- CDK specific fields -->
   <petalsCDK:mep>InOut</petalsCDK:mep>
   <petalsCDK:operation>${input_operation}</petalsCDK:operation>

   <!-- SOAP specific fields -->
   <soap:address>RESTService</soap:address>
   <soap:mode>REST</soap:mode>
   <soap:rest-add-namespace-uri>http://petals.ow2.org/sample</soap:rest-add-namespace-uri>
   <soap:rest-add-namespace-prefix>ns1</soap:rest-add-namespace-prefix>
   <soap:rest-remove-prefix-on-response>*</soap:rest-remove-prefix-on-response>
  </jbi:consumes>
 </jbi:services>
</jbi:jbi>
```

Each request to the REST URI `http://<host>:<port>/petals/services/RESTService/` will be forwarded to the SampleProviderEndpoint JBI endpoint.

An incoming request on the URL `http://<host>:<port>/petals/services/RESTService/myOperation?param1=value1&param2=value2` in GET mode will produce the following JBI message :

```
<myOperation>
  <param1>value1</param1>
  <param2>value2</param2>
</myOperation>
```

With the `rest.add-namespace-*` parameters specified in the previous configuration, the JBI message will be like :

```
<ns1:myOperation xmlns:ns1="http://petals.ow2.org/sample">
  <ns1:param1>value1</ns1:param1>
  <ns1:param2>value2</ns1:param2>
</ns1:myOperation>
```

Let's suppose that the JBI service returns a JBI response like :

```
<ns1:Response xmlns:ns1="http://petals.ow2.org/sample1" xmlns:ns2="http://petals.ow2.org/sample2"
 xmlns:ns3="http://petals.ow2.org/sample3">
  <ns1:param1>value1</ns1:param1>
  <ns2:param2>value2</ns2:param2>
  <ns3:param3>value3</ns3:param3>
</ns1:Response>
```

If the `rest.remove-prefix-on-response` parameter is set to 'ns1,ns2', the message returned to the REST service consumer will be :

```
<Response xmlns:ns1="http://petals.ow2.org/sample1" xmlns:ns2="http://petals.ow2.org/sample2"
 xmlns:ns3="http://petals.ow2.org/sample3">
  <param1>value1</param1>
  <param2>value2</param2>
  <ns3:param3>value3</ns3:param3>
</Response>
```

If the `rest.remove-prefix-on-response` parameter is set to '*', the message returned to the REST service consumer will be :

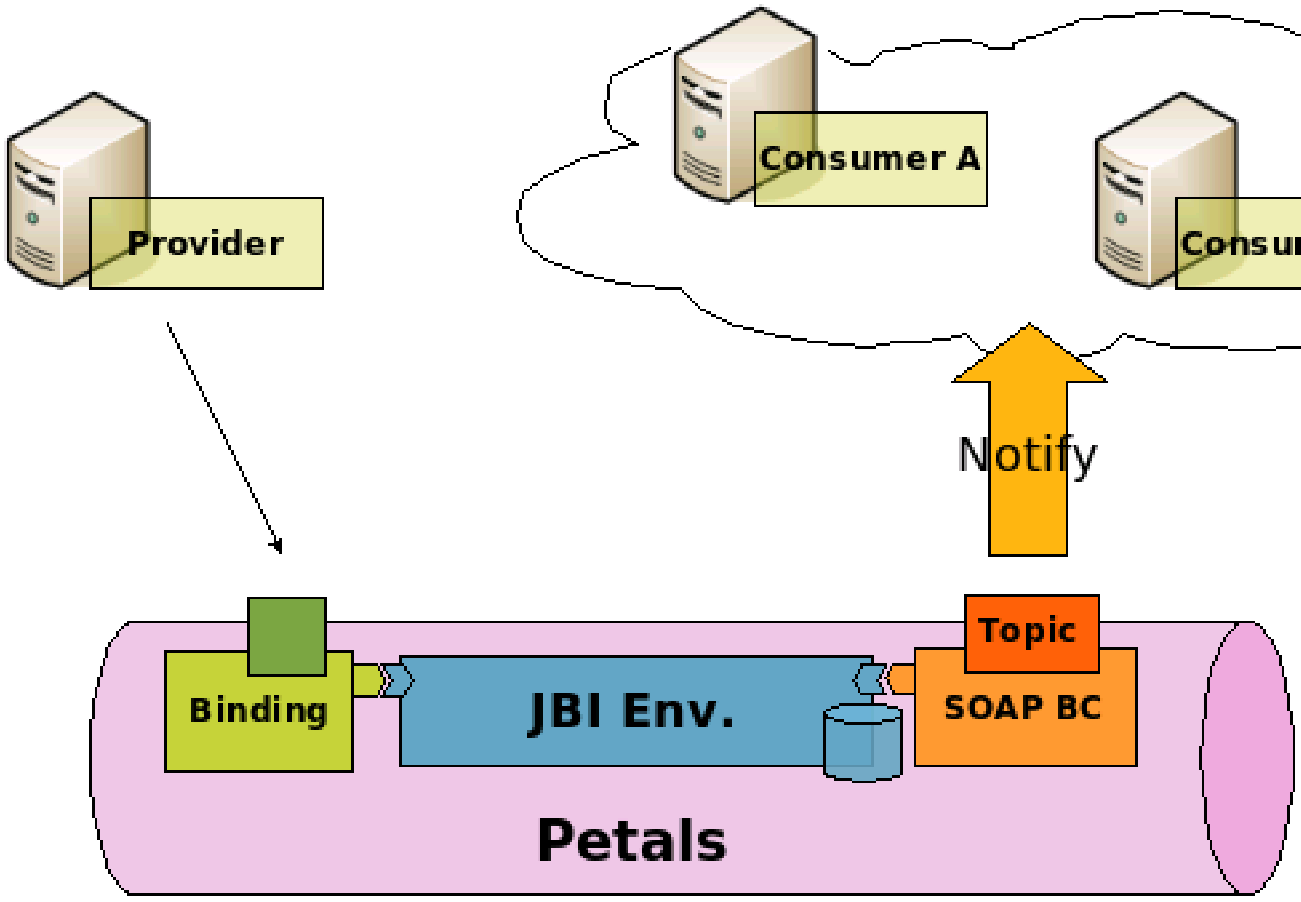
```
<Response xmlns:ns1="http://petals.ow2.org/sample1" xmlns:ns2="http://petals.ow2.org/sample2"
 xmlns:ns3="http://petals.ow2.org/sample3">
  <param1>value1</param1>
  <param2>value2</param2>
  <param3>value3</param3>
</Response>
```

Chapter 5. Web Service Notifications

5.1. Intoduction

The petals-bc-soap offers a Web Service Notification feature. It works as :

Figure 5.1. Handling Web Service notifications



WS-N is a family of related specifications that define a standard Web Service approach to notification using a topic-based publish/subscribe pattern. You can get the WS-N specification here. The SOAP binding component uses the petals-ws-star library to provide this feature.

As defined in the WS-N specification, each notification consumer must subscribe to the notification producer to receive notification messages. In PEtALS, a topic is linked to a JBI endpoint. Each time that a message is received on this endpoint, a notification message will be sent to notification WS consumers (see Figure 5.1, “Handling Web Service notifications”).

5.2. Create a WS-N topic

To create a WS-N topic, you need to deploy a service unit with a specific address format:

```
<?xml version="1.0" encoding="UTF-8"?>
<jbi:jbi version="1.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:jbi="http://java.sun.com/xml/ns/jbi
 xmlns:petalsCDK="http://petals.ow2.org/components/extensions/version-4.0"
 xmlns:soap="http://petals.ow2.org/components/soap/version-3.1"
 xmlns:sample="http://petals.ow2.org/soap/sample">

 <!-- Import a Service into PEtALS or Expose a PEtALS Service => use a BC. -->
 <jbi:services binding-component="true">

  <!-- Import a Service into PEtALS => provides a Service. -->
  <jbi:provides
   interface-name="sample:TopicInterface"
   service-name="sample:TopicService"
   endpoint-name="TopicEndpoint">

   <!-- SOAP specific fields -->
   <soap:address>TopicSample</soap:address>
   <soap:mode>TOPIC</soap:mode>1
   ...
  </jbi:provides>
 </jbi:services>
</jbi:jbi>
```

1 With the TOPIC mode, the topic specified in the address parameter will be created during Service Unit startup.

After deployment, a new JBI endpoint is available : TopicEndpoint. Each JBI message sent to this endpoint will be published on the topic. A WS-N producer is automatically created. It is in charge of handle the topic and send notification messages to all subscribers.

Note

The topics are persisted by the component since the release 3.1 so that all the required data is reloaded on restart.

5.3. Subscribe to WS-N producer

In order to receive WS-Notifications, the consumers MUST subscribe to these notifications to the WS-N producer.

To subscribe to WS notification, the notification consumer must send a specific SOAP message to the notification producer. In the SOAP BC, subscription URL is `http://HOST:PORT/wsn/producer` where :

- HOST is the host you have installed the SOAP BC
- PORT is the port where the SOAP BC listens to incoming SOAP messages

An example of a SOAP subscribe message is :

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
    <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
      http://localhost:8084/wsn-consumer/services/consumer
    </wsa:To>
    <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
      http://docs.oasis-open.org/wsn/bw-2/NotificationProducer/SubscribeRequest
    </wsa:Action>
    <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
      uuid:9888fa43-281f-ea0f-ec21-09e9119366c6
    </wsa:MessageID>
    <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
      <wsa:Address>http://www.w3.org/2005/08/addressing/role/anonymous</wsa:Address>
    </wsa:From>
  </soap:Header>

  <soap:Body>
    <wsnt:Subscribe xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2">
      <wsnt:ConsumerReference>
        <wsa:Address xmlns:wsa="http://www.w3.org/2005/08/addressing">
          http://127.0.0.1:8084/wsn-consumer/services/consumer1
        </wsa:Address>
      </wsnt:ConsumerReference>
      <wsnt:Filter>
         <wsnt:TopicExpression Dialect="xsd:anyURI">TopicSample</wsnt:TopicExpression>2
      </wsnt:Filter>
    </wsnt:Subscribe>
  </soap:Body>
</soap:Envelope>
```

1 The address to send notifications messages to. This can be simply a Web Service endpoint which can handle notification message

2 The name of the topic to subscribe to

Subscribers can use the PEtALS WS-N client API to subscribe to topics. It can be done like this :

```
package org.ow2.petals.binding.soap.wsn;

import java.net.URI;

import javax.xml.namespace.QName;

import org.ow2.petals.ws.addressing.EndpointReference;
import org.ow2.petals.ws.client.SubscriptionClient;
import org.ow2.petals.ws.client.WsnProducerClient;
import org.ow2.petals.ws.fault.WsnFault;
import org.ow2.petals.ws.notification.TopicExpressionFilter;

/**
 * Web service notification subscription.
 *
 */
public class SubscribeClient {

    /**
     * @param args
     */
    public static void main(String[] args) {

        EndpointReference sourceEPR = new EndpointReference(URI
                .create("http://localhost:9090/wsn-consumer/"));
        EndpointReference destinationEPR = new EndpointReference(URI
                .create("http://localhost:9090/wsn-consumer/service/consumer"));

        WsnProducerClient client = new WsnProducerClient(sourceEPR,
                destinationEPR);

        TopicExpressionFilter filter = null;
        try {
            filter = new TopicExpressionFilter(new QName("topicSample"));
        } catch (WsnFault e1) {
            e1.printStackTrace();
        }

        SubscriptionClient subsClient = null;
        try {
            subsClient = client.subscribe(sourceEPR, filter, null);
        } catch (WsnFault e) {
            e.printStackTrace();
        }
    }
}
```

Note

All the subscriptions are persisted in the component work folder to be able to reload subscriptions on component restart.

Note

If there are N subscriptions for the same notification consumer, the notification message will be sent N times. The first unsubscribe call will remove all the subscriptions for this consumer.

5.4. Send a WS notification from a JBI message

When the petals-bc-soap component receives a JBI message on a topic-activated endpoint, it is transformed into a WS notification message and published on the linked topic.

As an example of SOAP notification message, if the JBI message payload is :

```
<text>This is a sample of JBI message payload...</text>
```

and if it is published on the '**TopicSample**' topic, the SOAP body payload of the notification message will be :

```
<wsnt:Notify>
  <wsnt:NotificationMessage>
    <wsnt:SubscriptionReference>
      <wsa:Address xmlns:wsa="http://www.w3.org/2005/08/addressing">
        http://127.0.0.1:8084/wsn-consumer/services/consumer
      </wsa:Address>
    </wsnt:SubscriptionReference>
    <wsnt:Topic Dialect="xsd:anyURI">TopicSample</wsnt:Topic>
    <wsnt:ProducerReference>
      <wsa:Address xmlns:wsa="http://www.w3.org/2005/08/addressing">
        http://127.0.0.1:8084/wsn-producer/services/producer
    </wsa:Address>
    </wsnt:ProducerReference>
    <wsnt:Message>
      <text>This is a sample of JBI message payload...</text>
    </wsnt:Message>
  </wsnt:NotificationMessage>
</wsnt:Notify>
```

Chapter 6. Security

6.1. Introduction

The SOAP binding component provides WS security features through the Axis2 rampart module (http://ws.apache.org/axis2/modules/rampart/1_3/security-module.html).

This module is based on Apache WSS4J (http://ws.apache.org/wss4j), an implementation of the OASIS WS-security (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss).

This module is natively provided by the binding component since the 3.0 release.

6.2. Configuration

In order to enable WS-security, you must add specific extensions to the consumes section of the Service Unit. This configuration will tell Rampart which security mode to be applied. Here's an example of a jbi.xml providing a simple Rampart configuration, with UsernameToken and Timestamping authentification :

```
<?xml version="1.0" encoding="UTF-8"?>
<jbi:jbi version="1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:jbi="http://java.sun.com/xml/ns/jbi"
 xmlns:petalsCDK="http://petals.ow2.org/components/extensions/version-4.0"
 xmlns:helloworld="http://petals.ow2.org/helloworld"
 xmlns:soap="http://petals.ow2.org/components/soap/version-3.1">

 <!-- Import a Service into PEtALS or Expose a PEtALS Service => use a BC. -->
 <jbi:services binding-component="true">

  <!-- Expose a PEtALS Service => consumes a Service. -->
  <jbi:consumes interface-name="helloworld:Helloworld" service-name="helloworld:HelloworldService"
 endpoint-name="HelloworldEndpoint">

   <!-- CDK specific fields -->
   <petalsCDK:mep>InOut</petalsCDK:mep>

   <!-- SOAP specific fields -->
   <soap:address>UserPasswordSecuredService</soap:address>
   <soap:remove-root>false</soap:remove-root>
   <soap:mode>SOAP</soap:mode>
   <soap:modules>rampart</soap:modules>
   <soap:service-parameters>
    <![CDATA[
     <parameter name="InflowSecurity">
      <action>
       <items>UsernameToken Timestamp</items>

 <passwordCallbackClass>org.ow2.petals.usecase.soapsecurity.handler.RawCBHandler</
passwordCallbackClass>
      </action>
     </parameter>
    ]]>
   </soap:service-parameters>
  </jbi:consumes>
 </jbi:services>
</jbi:jbi>
```

On this example, an Axis2 service will be created (MyExampleService) and is secured by a defined security handler:

- The **<soap:modules>rampart</soap:modules>** tag allows to engage the rampart module for the UserPasswordSecuredService service.
- The **<soap:service-parameters>** tag allows to configure rampart for this service, using the InflowSecurity parameter (you can also use the OutflowSecurity parameter).

The `org.ow2.petals.usecase.soapsecurity.handler.RawCBHandler` Class is the handler used by the service. The following code snippet is an example of Handler implementation to validate user/password credentials:

```
package org.ow2.petals.usecase.soapsecurity.handler;

import org.apache.ws.security.WSPasswordCallback;

import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;

import java.io.IOException;

public class RawCBHandler implements CallbackHandler {

    public void handle(Callback[] callbacks) throws IOException,
            UnsupportedCallbackException {
        for (int i = 0; i < callbacks.length; i++) {
            WSPasswordCallback pwcb = (WSPasswordCallback)callbacks[i];
            String id = pwcb.getIdentifer();
            if("bob".equals(id)) {
                pwcb.setPassword("bobPW");
            }
        }
    }
}
```

This class MUST be in the service classloader, the easiest way is to package it in the service unit. It will be handled by the SOAP binding component and the Rampart module.

Note

If you use maven2 to package you service unit, you just have to add this java class under a `src/main/java` directory of your jbi-service-unit project or add a dependency to the handler library.

The service is now secured with Rampart. If a SOAP message without security headers is handled by the service, a SOAP fault will be returned with message like: *"Incoming message does not contain required Security header"*.

6.3. Client side

The SOAP header must contains the required security elements like in the following SOAP message snippet :

```
<soapenv:Header>
   <wsse:Security
    xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
    soapenv:mustUnderstand="1">
      <wsu:Timestamp

 xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
       wsu:Id="Timestamp-26598747">
         <wsu:Created>2007-07-30T14:59:34.944Z</wsu:Created>
         <wsu:Expires>2007-07-30T15:04:34.944Z</wsu:Expires>
      </wsu:Timestamp>
      <wsse:UsernameToken

 xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
       wsu:Id="UsernameToken-6427893">
         <wsse:Username>bob</wsse:Username>
         <wsse:Password

 Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-
1.0#PasswordDigest">
            0ziDIJ4Gd0XHbbbB/rgasDpOZJY=
         </wsse:Password>
         <wsse:Nonce>
            fqgz0lkb7/ezFiY7Km4qvg==
         </wsse:Nonce>
         <wsu:Created>
            2007-07-30T14:59:34.944Z
         </wsu:Created>
      </wsse:UsernameToken>
   </wsse:Security>
</soapenv:Header>
```

The following code snippet shows how to engage the rampaet module on the client side and how to call the Web Service :

```
ConfigurationContext ctx = ConfigurationContextFactory
  .createConfigurationContextFromFileSystem(axis2ConfPath, null);

ServiceClient client = new ServiceClient(ctx, null);
OMElement payload = getSayHelloOMElement(sayHelloStr);

Options options = new Options();
options.setProperty(WSSHandlerConstants.OUTFLOW_SECURITY, getOutflowConfiguration("bob"));
client.engageModule(new QName("rampart"));

options.setTo(targetEPR);
options.setAction("sayHello");

client.setOptions(options);
result = client.sendReceive(payload);
```

The `axis2ConfPath` directory must point to a directory in which a `modules` directory contains the `rampart-1.2.mar` module used by the client. The code also uses a Class handler which is similar to the service's one, and will provide the required user and password :

```
package org.ow2.petals.security.client.handler;

import org.apache.ws.security.WSPasswordCallback;

import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;

import java.io.IOException;

public class MyExampleClientHandler implements CallbackHandler {

    public void handle(Callback[] callbacks) throws IOException,
            UnsupportedCallbackException {
        for (int i = 0; i < callbacks.length; i++) {
            WSPasswordCallback pwcb = (WSPasswordCallback)callbacks[i];
            String id = pwcb.getIdentifer();
            if("bob".equals(id)) {
                pwcb.setPassword("bobPW");
            }
        }
    }
}
```

In this example, the user name is sent in **plain clear text** in the request. Depending on your security needs, you should use a secured transport layer (such as HTTPS), or another Rampart configuration to encrypt the information (and even the body content if required). For more Rampart configuration examples, you should have a look at the samples provided by Apache in the rampart distribution at : http://www.apache.org/dyn/closer.cgi/ws/rampart/1_3.

Chapter 7. Samples

The SOAP binding component samples are available as packaged use cases. You can find them in the download section of the project http://petals.ow2.org/downloads.html#usecases.