



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>

- March 2008 -



(CC) EBM WebSourcing - This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/3.0/>



Table of Contents

PETALS-BC-SOAP	5
1. Features	6
2. Component Configuration	7
3. Service Configuration	9
3.1. Send a JBI message to an external Web Service	9
3.1.1. Service Unit descriptor	9
3.2. Send a JBI message from an incoming SOAP message	10
3.2.1. Service Unit descriptor	11
4. REST Services	14
4.1. Introduction	14
4.2. Configuration	14
4.2.1. Provide mode : Provide access to external REST Service	14
4.2.2. Consume mode : Expose JBI Service as as REST Service	15
4.3. Sample	15
5. Web Service Notifications	17
5.1. Introduction	17
5.2. Create a WS-N topic	17
5.3. Subscribe to WS-N producer	18
5.4. Send a WS notification from a JBI message	19
6. Security	21
6.1. Introduction	21
6.2. Configuration	21
6.3. Client side	22

List of Figures

3.1. Provides an external Web Service as a JBI service	9
3.2. Consumes a JBI service on SOAP message	11
5.1. Handling Web Service notifications	17

List of Tables

2.1. Component installation configuration attributes	8
2.2. Advanced configuration of the component	8
3.1. Service Unit attributes to provide services	10
3.2. service-unit attributes to consume services	12
3.3. Advanced configuration of Service Unit (consumes elements)	13

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 v3.0, [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 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/jbi ../schema/jbi/standardPetals.xsd"
  xmlns:petals="http://petals.objectweb.org/extensions"
  xmlns:jbi="http://java.sun.com/xml/ns/jbi"
  xmlns:foo="http://www.foo.com/ns/bar" version="1.0">
  <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 />❶
    <jbi:bootstrap-class-name>org.ow2.petals.binding.soap.SoapBootstrap</jbi:bootstrap-class-name>
    <jbi:bootstrap-class-path/>❷

    <petals:pool-size>10</petals:pool-size>
    <petals:ignored-status></petals:ignored-status>

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

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

    <petals:component-interceptors/>

    <petals:properties-file/>

    <petals:params>
      <petals:param name="http.port">8084</petals:param>
      <petals:param name="http.services.list">true</petals:param>
      <petals:param name="http.thread.pool.size.min">1</petals:param>
      <petals:param name="http.thread.pool.size.max">255</petals:param>
      <petals:param name="http.acceptors">4</petals:param>
    </petals:params>
  </jbi:component>
</jbi:jbi>
```

- ❶ The component classpath element is filled during the component generation
- ❷ The bootstrap classpath element is filled during the component generation

Warning

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

This component is also configurable through JMX during its installation phase. Please refer to the JMX configuration manual for more information.

Table 2.1. Component installation configuration attributes

Attribute	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	true	No
http.thread.pool.size.minimum	Minimum size of the Jetty HTTP server thread pool	1	No
http.thread.pool.size.maximum	Maximum size of the Jetty HTTP server thread pool	255	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

More information about Jetty tuning can be found [here](#).

Table 2.2. Advanced configuration of the component

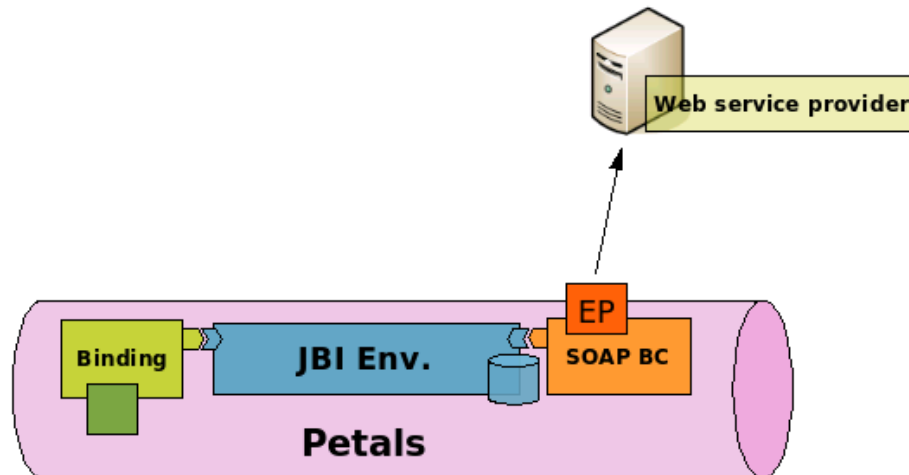
Parameter	Description	Default	Required
pool-size	Number of threads listening to messages coming from the JBI container (JBIListeners). Int number >= 1	10	No
ignored-status	Status of messages exchanges that component must ignore. Accepted values : DONE_AND_ERROR_IGNORED, DONE_IGNORED, ERROR_IGNORED or NOTHING_IGNORED	DONE_AND_ERROR_IGNORED	NO_IGNORED
jbi-listener-class-name	Fully qualified name of the class extending AbstractJBIListener		Yes
external-listener-class-name	Fully qualified name of the class extending AbstractExternalListener		No
properties-file	Name of the file containing values of keys used as reference by other parameters. To be able to configure a service-unit, you will use a key that has its value hosted by the component (ie. CDK documentation). The value of this parameter is : <ul style="list-style-type: none"> • whether an URL, • or a file relative to the directory defined by the environment variable PETALS_HOME. 		No

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:soap="http://petals.ow2.org/extensions"
  xmlns:jbi="http://java.sun.com/xml/ns/jbi"
  xmlns:petals="http://petals.ow2.org/">
```

```

<jbi:services binding-component="true">
  <jbi:provides interface-name="soap1"
    service-name="soap1"
    endpoint-name="soap1">
    <soap:wSDL>http://localhost:8080/axis2/services/HelloWorldService?wsdl</soap:wSDL>

    <soap:params>
      <soap:param name="address">http://localhost:8080/axis2/services/HelloWorldService</soap:param>
      <soap:param name="mode">SOAP</soap:param>
    </soap:params>

  </jbi:provides>
</jbi:services>
</jbi:jbi>

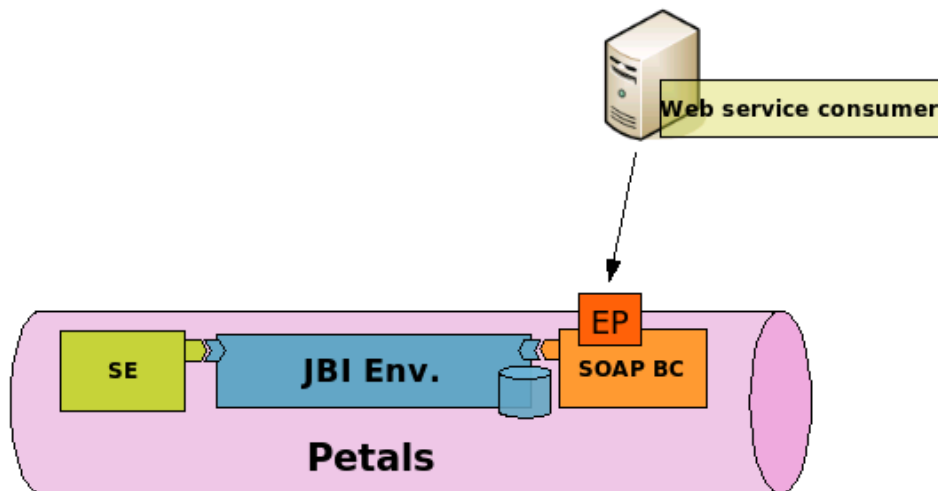
```

Table 3.1. Service Unit attributes to provide services

Attribute	Description	Default	Required
provides	Name of the JBI service that will be activated to expose the Web Service into the JBI environment. interface (qname), service (qname) and endpoint (string) name are required.		Yes
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 to be used to create SOAP messages. Possible values are "11" and "12".	11	No
timeout	The timeout value. Client will time out after waiting this amount of time. The value is expressed in milliseconds.	2000	No
proxy-host	The proxy host name. If it has not been set, the proxy mode will be disabled.		No
proxy-port	The proxy host port.	-1	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.	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 actually 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:soap="http://petals.ow2.org/extensions"
  xmlns:jbi="http://java.sun.com/xml/ns/jbi"
  xmlns:petals="http://petals.ow2.org/">
```

```

<jbi:services binding-component="true">
  <jbi:consumes interface-name="soap1"
    service-name="soap1"
    endpoint-name="soap0">

    <soap:mep>InOut</soap:mep>
    <soap:operation>sayHello</soap:operation>
    <soap:timeout>0</soap:timeout>

    <soap:params>
      <soap:param name="address">myWS</soap:param>
    </soap:params>

  </jbi:consumes>
</jbi:services>
</jbi:jbi>

```

SOAP communication attributes :

Table 3.2. service-unit attributes to consume services

Attribute	Description	Default	Required
consumes	Name of the JBI service that will be called into the JBI environment. Only the interface (qname) name can be provided (the container will choose a ServiceEndpoint for this interface), or you can only set service (qname) and endpoint (string) names, without the interface name.		Yes
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.		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 for created Axis2 service.		No

Table 3.3. Advanced configuration of Service Unit (consumes elements)

Parameter	Description	Default	Required
mep	Message exchange pattern abbreviation. This parameter can be user in conjunction with a 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		Yes
operation	Operation to call on a service. This parameter can be used in conjunctcion with the sendXXX methods of the Listeners. If no operation is specified in the MessageExchange to send, this parameter will be used.		Yes
timeout	Timeout in milliseconds in a synchronous send. this parameter can be used in conjunction with the sendSync(Exchange exchange) method of the Listeners. With this, a synchronous send is done with this timeout value. 0 for no timeout int number > 0 for a timeout	0	No
org.ow2.petals.routing.strategy	Used only in platform (distributed) PEtALS distribution. Two kind of strategy can be defines: highest or random. The others parameters represents respectively the local ponderation, the ponderation of the remote active endpoint and the ponderation of the remote inactive endpoint. The 'random' strategy chooses an endpoint in function of defined ponderations. The endpoints that have the strongest ponderation can be more easily choose in comparison with the others. The 'highest' strategy chooses the first endpoint in the list that have the strongest ponderation.		No
org.ow2.petals.transport.compress	Used only in platform (distributed) PEtALS distribution. When large and redondant information is contained in the payload of a MessageExchange, it can be interesting to compress the payload to reduce the volume of data to transfer between two PEtALS nodes. Values are true or false. True activated the compression of the messages payload.	false	No
org.ow2.petals.messages.ignoreDoneOrError	Used only in platform (distributed) PEtALS distribution. All MessageExchange and only a message containing a DONE OR ERROR status. The consumer must accept those messages, otherwise they are accumulated in the NMR. With this parameter , they can be ignored to reduce the PEtALS bus traffic. Possible values are true or false. Setting a true value makes the PEtALS container ignoring acknowledgment messages addressed to the deployed SU. This feature is unactivated when using synhronous sends.		No
org.ow2.petals.transport.policy	Used only in platform (distributed) PEtALS distribution. This property set up the policy of the Quality of Service supported by Petals Transporter. Possible values are : reliable, fast. If not specified, the reliable policy is selected by default.		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](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:petals="http://petals.ow2.org/extensions">

  <jbi:services binding-component="true">
    <jbi:provides interface-name="petals:Interface"
      service-name="petals:Service"
      endpoint-name="Endpoint">
      ...
      <petals:params>
        <petals:param
          name="address">http://example.org/param1={xpathexpression1}&param2={xpathexpression1}</
        petals:param>
        <petals:param name="mode">REST</petals:param>
        <petals:param name="http-method">GET</petals:param>
      </petals:params>

    </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 address 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 invocation.
 - 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

```
<?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:petals="http://petals.ow2.org/extensions">

  <jbi:services binding-component="true">
    <jbi:consumes interface-name="petals:Interface"
      service-name="petals:Service"
      endpoint-name="Endpoint">

      <petals:mep>InOut</petals:mep>
      <petals:operation>getXXX</petals:operation>
      ...
      <petals:params>
        <petals:param name="address">RESTServiceName</petals:param>
        <petals:param name="mode">REST</petals:param>
      </petals:params>

    </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.

4.3. Sample

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:petals="http://petals.ow2.org/extensions">

  <jbi:services binding-component="true">
    <jbi:provides interface-name="petals:YahooWeatherInterface"
      service-name="petals:YahooWeatherService"
      endpoint-name="YahooWeatherEndpoint">
      <petals:wSDL>Weather.wsdl</petals:wSDL>

      <petals:params>
        <petals:param
          name="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]}</
          petals:param>
        <petals:param name="mode">REST</petals:param>
        <petals:param name="http-method">GET</petals:param>
      </petals:params>

    </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 with the following JBI message :

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

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

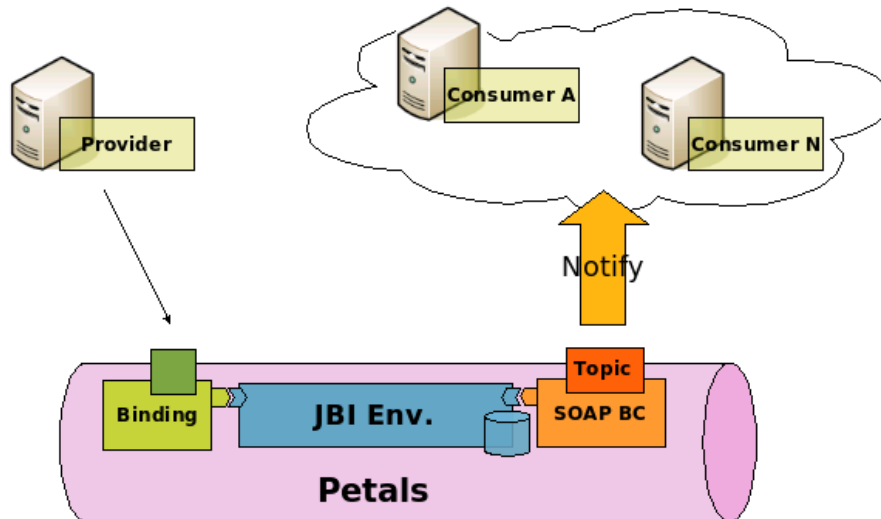
      <link>http://us.rd.yahoo.com/dailynews/rss/weather/Toulouse__FR/*http://weather.yahoo.com/forecast/FRXX0099_c.html</link>
      <pubDate>Thu, 06 Mar 2008 3:00 pm CET</pubDate>
      <yweather:condition code="28" date="Thu, 06 Mar 2008 3:00 pm CET" temp="8" text="Mostly Cloudy" />
      <description>
        <![CDATA[<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>
```


Chapter 5. Web Service Notifications

5.1. Introduction

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:

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

  <jbi:services binding-component="false">
    <jbi:provides interface-name="petals:TopicInterface"
      service-name="petals:TopicService"
      endpoint-name="TopicEndpoint">

      <extensions:params>
        <extensions:param name="address">TopicSample</extensions:param>
        <extensions:param name="mode">TOPIC</extensions:param>
        ...
      </extensions:params>
    </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:NotificationMessage>
</wsnt:Notify>

```

```
</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 authentication :

```
<?xml version="1.0" encoding="UTF-8"?>

<jbi:jbi version="1.0" xmlns:jbi="http://java.sun.com/xml/ns/jbi"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:technical="http://myexamples.technical.services.esb.acoss.fr/1.0"
xmlns:extensions="http://petals.objectweb.org/extensions/"
xmlns:keyvalue="http://petals.objectweb.org/extensions/key-value/">
<jbi:services binding-component="true">
  <jbi:consumes interface-name="technical:MyExample"
    service-name="technical:MyExampleService">
    <extensions:params>
      <extensions:param name="address">MyExampleService</extensions:param>
      <extensions:param name="modules">rampart</extensions:param>
      <extensions:param name="service-parameters">
        <![CDATA[
          <parameter name="InflowSecurity">
            <action>
              <items>UsernameToken Timestamp</items>
            </action>
          </parameter>
        ]]>
      </extensions:param>
    </extensions:params>
  </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 `<extensions:param name="modules">rampart</extensions:param>` tag allows to engage the rampart module for the MyExampleService service.
- The `<extensions:param name="service-parameters">` tag allows to configure rampart for this service, using the InflowSecurity parameter (you can also use the OutflowSecurity parameter).

The `org.objectweb.petals.security.handler.MyExampleHandler` 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.security.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 MyExampleHandler 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 provided 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.

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 rampart 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.