SOAPClient

The SOAPClient Action supports consumption of externally hosted Webservice endpoints.

The SOAPClient Action uses the soapUI Client Service to construct and populate a message for the target service. This action then routes that message to that service.

Endpoint Operation Specification

Specifying the endpoint operation is a straightforward task. Simply specify the "wsdl" and "SOAPAction" properties on the SOAPClient action as follows:

```
<action name="soapui-client-action" class="org.jboss.soa.esb.actions.soap.SOAPClient">
    <property name="wsdl" value="http://localhost:18080/acme/services/OrderManagement?wsdl"></property>
    <property name="SOAPAction" value="http://www.acme.com/OrderManagement/SendSalesOrderNotification"></property>
</action>
```

The SOAP operation is derived from the SOAPAction.

SOAP Request Message Construction

The SOAP operation parameters are supplied in one of 2 ways:

1. As a Map instance set on the *default body location* (Message.getBody().add(Map))
2. As a Map instance set on in a *named body location* (Message.getBody().add(String, Map)), where the name of that body location is specified as the value of the "paramsLocation" action property.

The parameter Map itself can also be populated in one of 2 ways:

1. **Option 1**: With a set of Objects that are accessed (for SOAP message parameters) using the OGNL framework. More on the use of OGNL below.
2. **Option 2**: With a set of String based key-value pairs(<String, Object>), where the key is an OGNL expression identifying the SOAP parameter to be populated with the key's value. More on the use of OGNL below.

As stated above, OGNL is the mechanism we use for selecting the SOAP parameter values to be injected into the SOAP message from the supplied parameter Map. The OGNL expression for a specific parameter within the SOAP message depends on that the position of that parameter within the SOAP body. In the following message:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
            xmlns:cus="http://schemas.acme.com">
 <soapenv:Header></soapenv:Header>
 <soapenv:Body>
  <cus:customerOrder>
     <cus:header>
        <cus:customerNumber>123456</cus:customerNumber>
     </cus:header>
  </cus:customerOrder>
 </soapenv:Body>
</soapenv:Envelope&gt
```

The OGNL expression representing the **customerNumber** parameter is
"**customerOrder.header.customerNumber**".

Once the OGNL expression has been calculated for a parameter, this class will check the supplied parameter map for an Object keyed off the full OGNL expression (Option 1 above).
  If no such parameter Object is present on the map, this class will then attempt to load the parameter by supplying the map and OGNL expression instances to the OGNL toolkit (Option 2 above). If this doesn't yield a value, this parameter location within the SOAP message will remain blank.

Taking the sample message above and using the "Option 1" approach to populating the "customerNumber" requires an object instance (e.g. an "Order" object instance) to be set on the parameters map under the key "customerOrder". The "customerOrder" object instance needs to contain a "header" property (e.g. a "Header" object instance). The object instance behind the "header" property (e.g. a "Header" object instance) should have a "customerNumber" property.

OGNL expressions associated with Collections are constructed in a slightly different way. This is easiest explained

through an example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:cus="http://schemas.active-endpoints.com/sample/customerorder/2006/04/CustomerOrder.xsd"
        xmlns:stan="http://schemas.active-endpoints.com/sample/standardtypes/2006/04/StandardTypes.xsd">
    <soapenv:Header></soapenv:Header>
    <soapenv:Body>
       <cus:customerOrder>
          <cus:items>
             <cus:item>
                <cus:partNumber>FLT16100</cus:partNumber>
                <cus:description>Flat 16 feet 100 count</cus:description>
                <cus:quantity>50</cus:quantity>
                <cus:price>490.00</cus:price>
                <cus:extensionAmount>24500.00</cus:extensionAmount>
             </cus:item>
             <cus:item>
                <cus:partNumber>RND08065</cus:partNumber>
                <cus:description>Round 8 feet 65 count</cus:description>
                <cus:quantity>9</cus:quantity>
                <cus:price>178.00</cus:price>
                <cus:extensionAmount>7852.00</cus:extensionAmount>
             </cus:item>
          </cus:items>
       </cus:customerOrder>
    </soapenv:Body>
</soapenv:Envelope>
```

The above order message contains a collection of order "items". Each entry in the collection is represented by an "item" element. The OGNL expressions for the order item "partNumber" is constructed as "**customerOrder.items{FOOTNOTE DEF 0 0}.partnumber**" and "**customerOrder.items{FOOTNOTE DEF 1 1}.partnumber**". As you can see from this, the collection entry element (the "item" element) makes no explicit appearance in the OGNL expression. It is represented implicitly by the indexing notation. In terms of an Object Graph (Option 1 above), this could be represented by an Order object instance (keyed on the map as "customerOrder") containing an "items" list (java.util.List or array), with the list entries being "OrderItem" instances, which in turn contains "partNumber" etc properties.

Option 2 (above) provides a quick-and-dirty way to populate a SOAP message without having to create an Object model ala Option 1. The OGNL expressions that correspond with the SOAP operation parameters are exactly the same as for Option 1, except that there's not Object Graph Navigation involved. The OGNL expression is simply used as the key into the Map, with the corresponding key-value being the parameter.

SOAP Response Message Consumption

The SOAP response object instance can be is attached to the ESB Message instance in one of the following ways:

1. On the *default body location* (Message.getBody().add(Map))
2. On in a *named body location* (Message.getBody().add(String, Map)), where the name of that body location is specified as the value of the "responseLocation" action property.

The response object instance can also be populated (from the SOAP response) in one of 3 ways:

1. **Option 1**: As an Object Graph created and populated by the XStream toolkit.
2. **Option 2**: As a set of String based key-value pairs(<String, String>), where the key is an OGNL expression identifying the SOAP response element and the value is a String representing the value from the SOAP message.
3. **Option 3**: If Options 1 or 2 are not specified in the action configuration, the raw SOAP response message (String) is attached to the message.

Using XStream as a mechanism for populating an Object Graph (Option 1 above) is straightforward and works well, as long as the XML and Java object models are in line with each other.

The XStream approach (Option 1) is configured on the action as follows:

```
<action name="soapui-client-action" class="org.jboss.soa.esb.actions.soap.SOAPClient">
   <property name="wsdl" value="http://localhost:18080/acme/services/OrderManagement?wsdl"></property>
   <property name="SOAPAction" value="http://www.acme.com/OrderManagement/GetOrder"></property>
   <property name="paramsLocation" value="get-order-params" ></property>
   <property name="responseLocation" value="get-order-response" ></property>
   <property name="responseXStreamConfig">
      <alias name="customerOrder" class="com.acme.order.Order" namespace="http://schemas.acme.com/services/CustomerOrde
      <alias name="orderheader" class="com.acme.order.Header" namespace="http://schemas.acme.com/services/CustomerOrder
      <alias name="item" class="com.acme.order.OrderItem" namespace="http://schemas.acme.com/services/CustomerOrder.xsd"
   </property>
</action>
```

In the above example, we also include an example of how to specify non-default named locations for the request parameters Map and response object instance.

We also provide, in addition to the above XStream configuration options, the ability

to specify field name mappings and XStream annotated classes.

```
<property name="responseXStreamConfig">
   <fieldAlias name="header" class="com.acme.order.Order" fieldName="headerFieldName" ></fieldAlias>
   <annotation class="com.acme.order.Order" ></annotation>
</property>
```

Field mappings can be used to map XML elements onto Java fields on those occasions when the local name of the element

does not correspond to the field name in the Java class.

To have the SOAP reponse data extracted into an OGNL keyed map (Option 2 above) and attached to the ESB Message, simply replace the "responseXStreamConfig" property with the "responseAsOgnlMap" property having a value of "true" as follows:

```
<action name="soapui-client-action" class="org.jboss.soa.esb.actions.soap.SOAPClient">
   <property name="wsdl" value="http://localhost:18080/acme/services/OrderManagement?wsdl"></property>
   <property name="SOAPAction" value="http://www.acme.com/OrderManagement/GetOrder"></property>
   <property name="paramsLocation" value="get-order-params" ></property>
   <property name="responseLocation" value="get-order-response" ></property>
   <property name="responseAsOgnlMap" value="true" ></property>
</action>
```

To return the raw SOAP message as a String (Option 3), simply omit both the "responseXStreamConfig" and "responseAsOgnlMap" properties.

HttpClient Configuration

The SOAPClient uses Apache Commons HttpClient to execute SOAP requests. It uses the HttpClientFactory to create and configure the HttpClient instance. For details on how to configure a HttpClient instance via the HttpClientFactory, see HttpClientFactory.

Specifying the HttpClientFactory configuration on the SOAPClient is very easy. Just add an additional property to the "wsdl" property as follows:

```
<property name="wsdl" value="https://localhost:18443/active-bpel/services/RetailerCallback?wsdl">
```

```
    <http-client-property name="file" value="/localhost-https-18443.properties" ></http-client-property>
  </property>
```

The "file" property value will be evaluated as a filesystem, classpath or URI based resource (in that order).

Properties may also be specified directly in the action configuration like so:

```
<property name="http-client-properties">
  <http-client-property name="http.proxyHost" value="localhost"/>
  <http-client-property name="http.proxyPort" value="8080"/>
</property>
```

Transforming the SOAP Request Message

It's often necessary to be able to transform the SOAP request before sending it. This may be to simply add some SOAP headers.

Transformation of the SOAP request (before sending) is supported by configuring the SOAPClient action with a Smooks transformation configuration property as follows:

```
  <property name="smooksTransform" value="/transforms/order-transform.xml" ></property>
```

The value of the "smooksTransform" property is resolved by first checking it as a filesystem based resource. Failing that, it's checked as a classpath resource and failing that, as a URI based resource.