Using REST with a Proxy Service

If you have a REST front-end client, REST back-end service, or both a REST client and service, you can use a proxy service in the ESB profile to handle the communication between the front end and back end. This page describes the following scenarios:

- REST Client and SOAP Service
- SOAP Client and REST Service
- REST Client and REST Service
- JMS Client and REST Service

REST Client and SOAP Service

This usecase represents the scenario where a REST front-end client has to communicate in plain old XML (POX) with a SOAP back-end service. This can be easily achieved by placing the WSO2 EI in the middle with a proxy service.

Assume the front-end REST client is using http/https as its transport protocol, and the back-end service is expecting SOAP1.1. The proxy service configuration would look like this:
In this scenario, the client sends a REST message to the ESB profile, which transforms the message to a SOAP1.1 message and sends it to the back-end service. Once the back-end response is received, the ESB profile sends the response back to the client as a REST message.

To implement this scenario:

1. Start the ESB profile and change the configuration as shown above.
2. Deploy the back-end service 'SimpleStockQuoteService' and start the Axis2 server using the instructions in Starting Sample Back-End Services.
   You will now send a message to the back-end service through the ESB profile using the sample Stock Quote Client. This client can run in several modes.
3. Run the following ant command from the <EI_HOME>/samples/axis2Client directory to trigger a sample message to the back-end service:
   ```
   ant stockquote -Daddurl=http://localhost:8280/services/StockQuoteProxy -Drest=true
   ```

If the message is mediated successfully, it will display an output on the Axis2 server's start-up console along with an output message on the client console. If you want to see exactly how the REST messages are transformed to SOAP messages, you could place an Apache tcpmon application between the client and WSO2 EI and another between WSO2 EI and the back-end service.

For another example of POX to SOAP conversion, see Sample 50: POX to SOAP Conversion.

**SOAP Client and REST Service**

This usecase represents the scenario where a SOAP front-end client has to communicate with a REST back-end service. This can be easily achieved by placing WSO2 EI in the middle with a proxy service.
The proxy service configuration would look like this:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<proxy xmlns="http://ws.apache.org/ns/synapse"
    name="CustomerServiceProxy"
    startOnLoad="true"
    statistics="disable"
    trace="disable"
    transports="http,https">
    <target>
        <inSequence>
            <filter xpath="//getCustomer">
                <then>
                    <property expression="//getCustomer/id"
                        name="REST_URL_POSTFIX"
                        scope="axis2"
                        type="STRING"/>
                    <property name="HTTP_METHOD" scope="axis2" type="STRING" value="GET"/>
                </then>
                <else/>
            </filter>
            <header name="Accept" scope="transport" value="application/xml"/>
            <send>
                <endpoint>
                    <address format="pox"
                        uri="http://localhost:9763/jaxrs_basic/services/customers/customerservice/customers"/>
                </endpoint>
            </send>
        </inSequence>
        <outSequence>
            <send/>
        </outSequence>
        <faultSequence/>
    </target>
    <description/>
</proxy>
```
In this scenario, the client sends a SOAP message to the ESB profile, which transforms it to a REST message and sends it to the back-end service. Once the back-end response is received, the ESB profile sends it back to the client as a SOAP message. The filter mediator is used to identify the required service. Though this example only has one filter, there could be many filters in the real scenario. The REST_URL_POSTFIX property is a well-defined property, and its value is appended to the target URL when sending messages out in a RESTful manner. Similar to REST_URL_POSTFIX, the HTTP_METHOD property is also well-defined and sets the HTTP verb to GET.

Note that the format of the endpoint is set to pox. This format works for GET, PUT, and DELETE methods, but if you want to use POST and the URI has a postfix, the format must be rest.

To implement this scenario:

1. Start the ESB profile and change the configuration as shown above.
2. Build and deploy the JAX-RS Basics sample. This sample is shipped with WSO2 EI. It contains a set of pure RESTful services, so we will use this as our back-end service.
3. Send a message to the back-end service through the ESB profile using SoapUI. The service used in this scenario is the getCustomer service, which requires the customer ID to complete its task. A sample SOAP message that is used to achieve this is as follows:

   ```xml
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
     <soapenv:Header/>
     <soapenv:Body>
       <getCustomer>
         <id>123</id>
       </getCustomer>
     </soapenv:Body>
   </soapenv:Envelope>
   ```

   This message simply represents the relevant method to be invoked and its associated value. Upon successful execution, SoapUI should display the following message in response.

   ```xml
   <Customer>
     <id>123</id>
     <name>John</name>
   </Customer>
   ```

   For another example, see Sample 152: Switching Transports and Message Format from SOAP to REST POX.

**REST Client and REST Service**

This usecase represents the scenario where a REST front-end client has to communicate with a REST back-end service. Even though they are both using the same architecture, let’s assume the front-end client is using JSON whereas the back-end service is using plain old XML (POX). This can be easily achieved by placing WSO2 EI in the middle with a proxy service.
The proxy service configuration is as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<proxy xmlns="http://ws.apache.org/ns/synapse"
    name="CustomerServiceProxy"
    startOnLoad="true"
    statistics="disable"
    trace="disable"
    transports="http,https">
  <target>
    <inSequence>
      <send>
        <endpoint>
          <address format="pox"
             uri="http://localhost:9763/jaxrs_basic/services/customerservice"/>
        </endpoint>
      </send>
    </inSequence>
    <outSequence>
      <send/>
    </outSequence>
    <faultSequence/>
  </target>
  <description/>
</proxy>
```

In this scenario, the ESB profile simply changes the message type of the client into XML and then passes it to the REST service. Once the ESB profile has received the XML message, it transforms it back into a JSON message and sends it to the client.

Typically, it’s very important to set the HTTP_METHOD property, because this represents the required HTTP action to be sent to the back-end service. However, in this scenario we will use a curl request, which is done using the HTTP GET method, so by default any request is using GET as its HTTP action, even if the HTTP_METHOD property is not set.

To implement this scenario:

1. Start the ESB profile and create a proxy service using the configuration shown above.
2. Build and deploy JAX-RS Basic sample. This contains a set of pure RESTful services, so we will use this as our back-end service.
3. Go to the command prompt and issue the following `curl` command to simulate the request of a real REST client:

   ```
   /CustomerServiceProxy/customers/123
   ```

   Following is the expected output that should appear on the console upon successful execution of the scenario:

   ```
   {
       "Customer":{
           "id":123,
           "name":"John"
       }
   }
   ```

**JMS Client and REST Service**

This use case represents the scenario where a JMS front-end client has to communicate with a REST back-end service. This can be easily achieved by placing WSO2 EI in the middle with a proxy service.

In this scenario, a proxy service in the ESB profile listens to a JMS queue, picks up available messages from that queue, and delivers the messages to the REST back-end service. For the JMS front-end client, we will use the sample JMSClient that ships with the ESB profile. For the REST back-end service, we will use the JAX-RS Basics sample service that is shipped with the ESB profile. We will use ActiveMQ as the message broker.

The proxy service configuration is as follows:

```xml
<proxy xmlns="http://ws.apache.org/ns/synapse" name="JmsToRestProxy" transports="jms" statistics="disable" trace="disable" startOnLoad="true">
  <target>
    <inSequence>
      <property name="OUT_ONLY" value="true"/>
      <send>
        <endpoint>
          <address uri="http://localhost:8280/jaxrs_basic/services/customers
          /customerservice/customers="/>
        </endpoint>
      </send>
    </inSequence>
  </target>
  <parameter name="transport.jms.ContentType">
    <rules>
      <jmsProperty>contentType</jmsProperty>
      <default>application/xml</default>
    </rules>
  </parameter>
  <description></description>
</proxy>
```

To implement this scenario:

1. Start ActiveMQ and configure the JMS transport in WSO2 EI to work with ActiveMQ.
2. Build and deploy the JAX-RS Basics sample.
3. Goto `<EI_HOME>/samples/axis2Client` and execute the following command:
   
   ```
   ant jmsclient -Djms_type=text -Djms_dest=dynamicQueues/JmsToRestProxy -
   Djms_payload="<Customer><name>WSO2</name></Customer>"
   ```

   This sends `<Customer><name>WSO2</name></Customer>` as the payload.

In the application server console, you will see the following line printed:

   ```
   ----invoking addCustomer, Customer name is: WSO2
   ```