

REST To SOAP Transformation in Mediator End to End tutorial

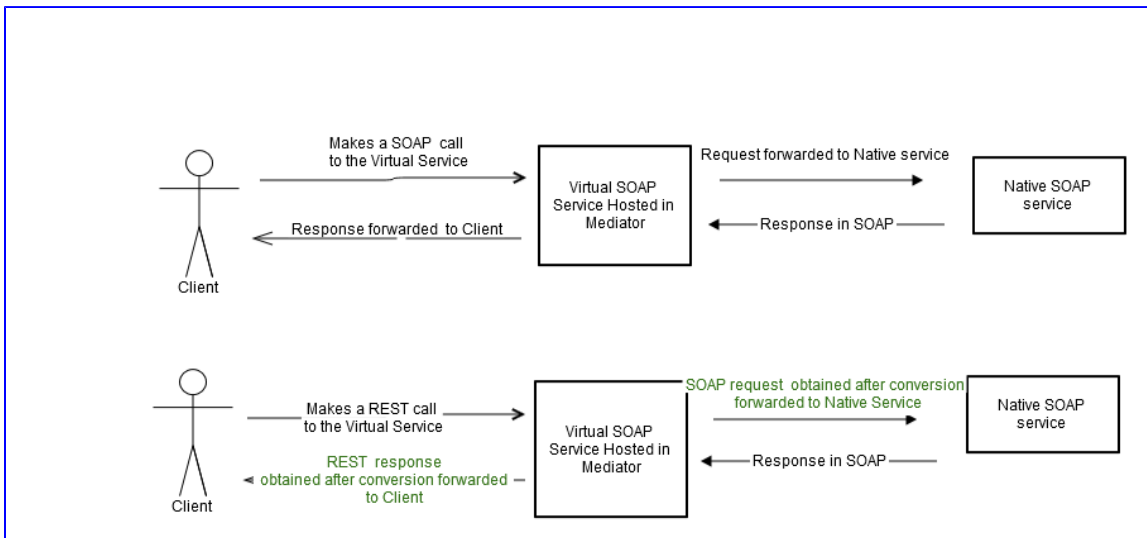
- Preface
 - End to End Flow
 - Enabling REST invokes for a Virtual SOAP Service:
 - SOAP - Version Handling:
 - Invoking a REST enabled SOAP Service:
 - SOAP Action Handling:
 - Response handling:
 - XSLT and ESB transformations:
 - Multipart-Form data handling:
-

Preface

This feature enables the customers to invoke all SOAP based services as REST services. Mediator Clients would now be able to make RESTful calls to Virtual SOAP services. Mediator would transform the REST request sent by the Client into SOAP requests as expected by the Native SOAP service. The transformation of SOAP response sent by the Native service into REST response is also done.

End to End Flow

The following diagram depicts the above use-case more in detail



Enabling REST invokes for a Virtual SOAP Service:

The provider can decide on what services he wants to expose for RESTful calls in Mediator. During virtualization he can use the "Enable REST Support" action present in the Request Handling step for this purpose.

Virtualize soap.addInts_WSD (Step 2 of 3)
 Configure the runtime aspects of your API by dragging policy actions and dropping them in the Message Flow area.

Policy Actions

Request Handling

- Protocol
- Request Transformation
- Invoke webMethods Integration Server
- Enable REST Support**
- Set Media type

Policy Enforcement

Response Processing

Error Handling

Message Flow

Receive

Require HTTP /HTTPS

Enable REST Support

Enforce

Operator (Applicable to only "Evaluate" actions) ☒ And ☐ Or

Routing

Straight Through Routing

Response

Note: This action is added by default during fresh virtualization of a service.

On publishing this service to Mediator the following changes are made in the VSD.

1. Mediator adds two HTTP bindings with methods "GET" and "POST" to the Virtual Service WSDL.
2. The VSD contains the "expose-as-rest" to indicate that this service has been enabled with a REST invoke.

```

<definition>
  <monitorPolicies />
  <inSequence>
    <expose-as-rest soap-version="soap11" />
    <http-config>
      <authentication mode="incoming" scheme="basicauth" />
    </http-config>
    <send>
      <endpoint>
        <address isAlias="false" optimize="none" passSecurityHeaders="false" uri="local://demo:multiplyInts_WSD">
          <connect-timeout>
            <duration>30</duration>
          </connect-timeout>
        </address>
      </endpoint>
    </send>
  </inSequence>

```

Note: It is not possible to have "Enable as REST" and WS-Security policies applied at the same time to a Virtual service. This is because of the restriction that REST invokes cannot have WSS headers. Please see XSLT and ESB transformations section on more details of how to support such a use-case.

SOAP - Version Handling:

Mediator needs to know the SOAP version of the Native Endpoint in order to transform the REST request it receives. This information is passed from the CentraSite which decides the SOAP version of the Endpoint selected from the Native service WSDL that is used for Virtualization and adds it to the VSD.

Invoking a REST enabled SOAP Service:

The REST endpoint for a SOAP service is nothing but the Operation name added to the Virtual Service Invoke path. I.e. `http(s) ://< HOST> :< PORT>/ws/<SERVICE-NAME>/<OPERATION-NAME>`.

Let's take an example of the StockQuote service (<http://www.websvcx.net/stockquote.asmx?WSDL>).

After virtualizing this service and publishing to Mediator it has 1 soap operation . Now the table explains the corresponding REST endpoint for the Operation

SOAP Action	REST -Endpoint for the Action
GetQuote	<IS-CONTEXT-URL>/StockQuoteService/GetQuote

The request can be of any one of the below content types:

- application/xml
- application/json
- application/json/badgerfish
- application/x-www-form-urlencoded
- text/xml
- multipart/form-data

Note : If No Content-type header is provided in the request Mediator default that the Request is of application/json type.

The below table now explains the possible REST requests to the above stock quote service. All requests are made to the Endpoint `http://<Context-path>/ StockQuoteService/ GetQuote` . The SOAP request is of the form

	Content-type	Request	Comments
1	application/xml text/xml	<code><web:GetQuote xmlns:web="http://www.webserviceX.NET/"> <web:symbol >IBM </web:symbol>< /web:GetQuote></code>	This is same to the SOAP Request without being wrapped over the SOAP envelope
2	application/xml	<code><web:symbol xmlns:web="http://www.webserviceX.NET/">IBM </web:symbol></code>	When the Operation Element is not present in the request , Mediator intelligently wraps the incoming request with the operation element
3	application/json	<code>{ "GetQuote" : { "symbol" : "IBM" } }</code>	Similar to 1 .Mediator adds the requires Namespaces to the converted requests. Hence it is not needed to send the Namespaces.
4	application/json	<code>{ "symbol" : "IBM" }</code>	Similar to 2
5	application/x-www-form-urlencoded	<code>?symbol=IBM</code>	This is passing the arguments in the URL encoded fashion.

Important Note for URL-Encoded parameters:

Mediator can support URL-Encoded parameters only for one level.
For example to a Native service that expects a SOAP request of the form

```
<add>
  <num1 />
  <num2 />
</add>
```

The above said XML have an equivalent URL -Encoded request like `http://<Context-path>/AddIntsService/add?num1=10&num2=9`

Whereas if the request has nested elements like

```
<add>
  <nums>
    <num1 />
    <num2 />
  </nums>
</add>
```

Then this CANNOT be supported with `application/x-www-form-urlencoded` content type.

SOAP Action Handling:

Mediator needs to add the SOAP-Action header to the converted SOAP request. It derives this information from the service WSDL.

```
- <wsdl:binding name="StockQuoteSoap12" type="tns:StockQuoteSoap">
  <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"/>
  - <wsdl:operation name="GetQuote">
    <soap12:operation soapAction="http://www.webserviceX.NET/GetQuote" style="document"/>
    - <wsdl:input>
      <soap12:body use="literal"/>
    </wsdl:input>
    - <wsdl:output>
      <soap12:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
```

Response handling:

Mediator during the Response phase should convert the Native service response (SOAP) to a REST response depending on the Accept header.

If no Accept Header is provided Mediator default it to application/json.

XSLT and ESB transformations:

When a SOAP service is enabled for REST invokes, a provider would need to configure XSLT's and ESB's only when the invoke is via a REST or vice versa.

For example there is a Native service that has WS-Security enforced in it .For SOAP invokes the client can add these data to the request to Mediator, which is not possible for REST invokes.

This can be achieved in Mediator with the help of a property pg_isRestInvoke. By using this property the provider can write his own XSLT's and ESB that change the details of Service invocation only if it is REST/SOAP.

A sample XSLT is shown below.

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl=" http://www.w3.org/1999/XSL/Transform" xmlns:
soapenv=" http://schemas.xmlsoap.org/soap/envelope/"
xmlns:esp=" http://bms.com/esp" version="1.0">

<xsl:param name="pg_isRestInvoke"/>

<xsl:template match="soapenv:Envelope">
  <xsl:choose>
    <xsl:when test="$pg_isRestInvoke = 'true'">
      <soapenv:Envelope>
        <soapenv:Header>
          <yes></yes>
        </soapenv:Header>
        <soapenv:Body>
          <!--Body Content goes here for REST-->
        </soapenv:Body>
      </soapenv:Envelope>
    </xsl:when>
    <xsl:otherwise>
      <soapenv:Envelope>
        <soapenv:Header>
          <no></no>
        </soapenv:Header>
        <soapenv:Body>
          <!--Body Content goes here for SOAP-->
        </soapenv:Body>
      </soapenv:Envelope>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
</xsl:stylesheet>

```

Note: The ESB and XSLT transformations are applied only after converting the Client's REST request to SOAP request. Hence the XSLT/ESB can modify the converted SOAP request as per requirements. .

Multipart-Form data handling:

The provider can have a Native SOAP service that expects attachments in the SOAP body. It can use any optimization technique (MTOM/SwA). The equivalent REST invoke to the service will be of type 'multipart/form-data'

Mediator also needs the optimization technique used for outbound request. This information is derived by Mediator from the Endpoint properties that is configured in CentraSite

The screenshot shows the 'Message Flow' configuration window on the left and the 'Endpoint Properties' dialog box on the right. The 'Message Flow' window has sections for 'Receive' (with 'Require HTTP / HTTPS' and 'Enable REST Support' checkboxes), 'Enforce' (with an 'Operator' dropdown set to 'And'), 'Routing' (with 'Straight Through Routing' selected), and 'Response'. The 'Endpoint Properties' dialog has a title bar with a close button. It contains several sections: 'SOAP Optimization Method' with a dropdown menu currently showing 'None' and a red arrow pointing to it; 'MTOM' with a list containing 'None' and 'SWA'; 'SSL Configuration' with 'Client Certificate Alias' and 'Keystore Alias' text boxes; and 'WS-Security Header' with a dropdown menu showing 'Remove processed security headers'. At the bottom of the dialog are 'Cancel' and 'OK' buttons. In the background, partially visible, are buttons for 'Previous', 'Virtualize', 'Next', and 'Cancel'.

Mediator uses the following rules to convert the multipart/form-data request to a request with SOAP attachments

- If the multipart/form-data requests contain an application/json part then the JSON part is converted and added as the SOAP body of the outgoing request.
- If an application/json part is not present Mediator checks for an application/xml part and adds it to the SOAP body.
- If no application/json or application/xml parts are found in the request an empty soap body with the operation element alone is added.
- If more than one application/json or application/xml parts are found in the request the first part is taken and processed. The remaining parts are sent as attachments to the outbound requests.

Consider the following Multipart Form Request.

Multipart/form-data request

```
POST http://localhost:5555/ws/EmployeeFileUploadServiceMTOM
/UploadFileRequest HTTP/1.1
Content-Type: multipart/form-data; boundary="-----=_Part_1_1421953406.
1422953553647"
MIME-Version: 1.0
Content-Length: 633
Host: localhost:5555
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

-----=_Part_1_1421953406.1422953553647
Content-Type: application/json; name=employeeDetails.json
Content-Transfer-Encoding: binary
Content-Disposition: form-data; name="employeeDetails.json"; filename="
employeeDetails.json"

{
  "employeeID" : "1",
  "filename" : "details.txt"
}

-----=_Part_1_1421953406.1422953553647
Content-Type: text/plain; charset=Cp1252; name=Tests.txt
Content-Transfer-Encoding: binary
Content-Disposition: form-data; name="data"; filename="Tests.txt"

TestPGSchemaUnwrapper 3 tests
TestJSONConverto 5 tests

1. native service
2. xslt

-----=_Part_1_1421953406.1422953553647--
```

The above request contains two parts

1. A JSON part with content-id 'employeeDetails.json'.
2. An text/plain part of content-id 'data'

Now the converted SOAP request is as shown below

Converted SOAP Request

```
POST /EmployeeDetail/services/employeeFile HTTP/1.1
User-Agent: Mozilla/4.0 [en] (WinNT; I)
Accept: image/gif, */*
Host: localhost:7022
Content-Type: multipart/related; boundary="
MIMEBoundary_e1283a82dcb388a3866f725c6809ae0871fad51b973cfa9b"; type="
application/xop+xml"; start="<0.
12283a82dcb388a3866f725c6809ae0871fad51b973cfa9b@apache.org>"; start-info="
application/soap+xml"; action="urn:uploadEmployeeFile"
MIME-Version: 1.0
Content-Length: 1142
```

```
--MIMEBoundary_e1283a82dcb388a3866f725c6809ae0871fad51b973cfa9b
Content-Type: application/xop+xml; charset=UTF-8; type="application
/soap+xml"
Content-Transfer-Encoding: binary
Content-ID: <0.12283a82dcb388a3866f725c6809ae0871fad51b973cfa9b@apache.org>
```

```
<?xml version='1.0' encoding='UTF-8'?><soapenv:Envelope xmlns:soapenv="
http://www.w3.org/2003/05/soap-envelope">
```

```
    <soapenv:Body>
        <axis2ns87:UploadFileRequest xmlns:axis2ns87="
http://mediator.softwareag.com/">
            <data><xop:Include xmlns:xop="http://www.w3.org
/2004/08/xop/include" href="cid:1.02283
a82dcb388a3866f725c6809ae0871fad51b973cfa9b@apache.org" />
            </data>
            <employeeID>1</employeeID>
            <filename>details.txt</filename>
        </axis2ns87:UploadFileRequest>
    </soapenv:Body>
</soapenv:Envelope>
```

```
--MIMEBoundary_e1283a82dcb388a3866f725c6809ae0871fad51b973cfa9b
Content-Type: text/plain; charset=Cp1252; name=Tests.txt
Content-Transfer-Encoding: binary
Content-ID: <1.02283a82dcb388a3866f725c6809ae0871fad51b973cfa9b@apache.org>
```

```
TestPGSchemaUnwrapper 3 tests
TestJSONConvertor 5 tests
```

1. native service
2. xslt

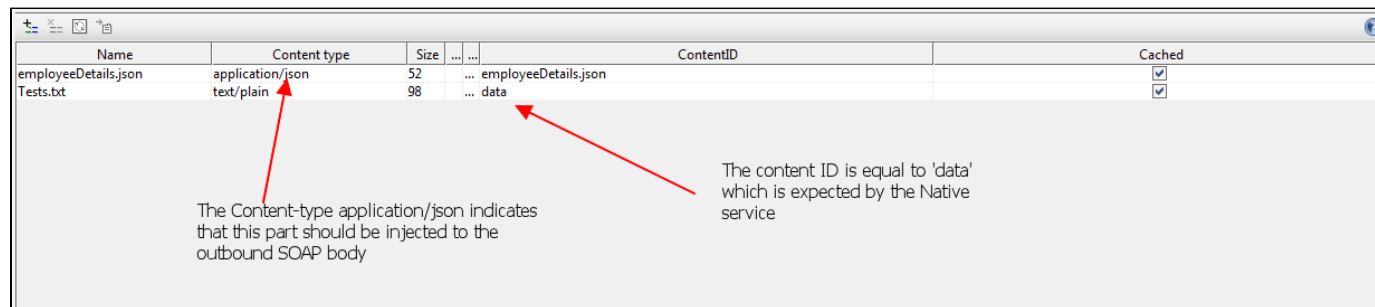
```
--MIMEBoundary_e1283a82dcb388a3866f725c6809ae0871fad51b973cfa9b--
```

Now the converted request has a SOAP Body (converted JSON part with ID 'employeeDetails.json'). Please note the

```
<data>
<xop:Include xmlns:xop="http://www.w3.org/2004/08/xop/include" href="cid:1.02283
a82dcb388a3866f725c6809ae0871fad51b973cfa9b@apache.org"/>
</data>
```

which is the attachment that is sent. Also the attachment XOP element is wrapped with the element 'data' which is the Content-ID of the attachment that is sent.

The below image shows the attachments in SOAPUI(Attachments tab in Request).



Name	Content type	Size	ContentID	Cached
employeeDetails.json	application/json	52	... employeeDetails.json	<input checked="" type="checkbox"/>
Tests.txt	text/plain	98	... data	<input checked="" type="checkbox"/>

The Content-type application/json indicates that this part should be injected to the outbound SOAP body

The content ID is equal to 'data' which is expected by the Native service