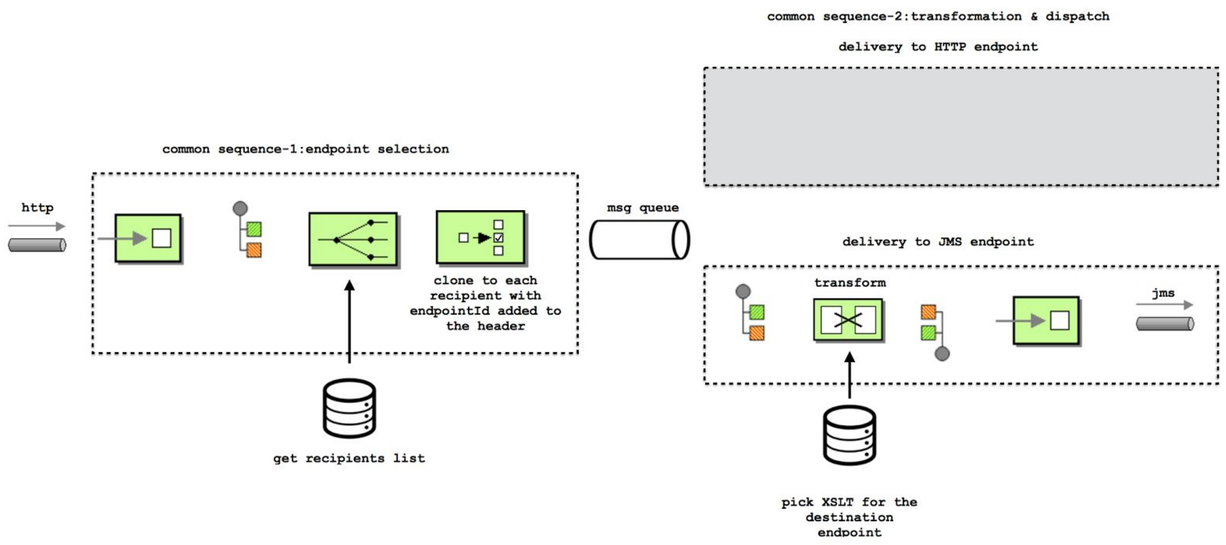
**About the Demo**

The following flow diagram explains the high level view of this demo


common sequence-2:transformation & dispatch
delivery to HTTP endpoint
common sequence-1:endpoint selection
http
msg queue
clone to each recipient with endpointId added to the header
delivery to JMS endpoint
transform
jms
get recipients list
pick XSLT for the destination endpoint

1. Messages can come from any of these 3 transport (HTTP/JMS/File) with a header (i.e ClientID). Three different proxy services will be used for this purpose as FileBasedListerner, HTTPBasedListener and JMSBasedListener

2. The incoming message from these listeners will then be received by a common sequence (EndPointSelectionSequence).

3. In this common sequence, based on the header (ClientId) a recipient list (an xml containing the endpoint Id’s) will be picked. For example, if the ClientID is “App1” then this sequence assumes that there will be xml file with the name “App1.xml” under a specified registry location.

4. After selecting the recipient list, and based on the number endpoints in the list, the message will be cloned with endpointId added as a JMS message property (EndPointId) and sent to a JMS message queue.

   For example, if the recipient list has two endpoint-Id’s (HttpEndpoint1 and JMSEndpoint1), then there will be two messages placed on the message queue with a JMS message property EndPointId, but the value of the property being HttpEndpoint1 and JMSEndpoint1 respectively.

5. A proxy service (CommonProxy) will listen to this JMS message queue and will then send that a sequence known as the CommonSequence.

6. In this common sequence, based on the JMS property (EndpointId) the transformer (XSLT mediator) will pick the correct XSLT and transform the message

7. And again based on the header (EndpointId) the router (Header Based Routing) will pick the correct endpoint from the endpoint list.

8. Then the transformed message will sent to the correct dispatcher. The outgoing dispatching can be also in any transports (JMS/HTTP/File)

## Running the Demo

1. Download WSO2 ESB Product from http://wso2.com/products/enterprise-service-bus/

2. Configure WSO2 ESB JMS transport with Apache ActiveMQ as described in : https://docs.wso2.com/display/ESB481/Configure+with+ActiveMQ

3. Configure WSO2 ESB VFS transport as described in : https://docs.wso2.com/display/ESB481/VFS+Transport

   Note : A sample axis2.xml with the default JMS transport configuration to connect to ActiveMQ and VFS transport configuration can be found under the resources/conf directory of the demo project source.

4. Checkout the demo project hosted in : https://github.com/Kishanthan/demos. This project consists of three maven modules as below.
   i. ESBDemoProject
   ii. ESBRegistryResourceDemo
   iii. CarbonAppProject

   The ESBDemoProject consists all the ESB related deployment artifacts (Proxies, Sequences). ESBRegistryResourceDemo project consists of all the artifacts that are deployed to registry (Endpoints, XSLT's and RecipientList xml resource). The CarbonAppProject project will pack both of these project and provide a single artifact as mentined in #6 below, which can be deployed on ESB.

5. Go to the **wso2-esb/header-based-routing** subdirectory and perform a maven clean build from that location. This will generate the necessary carbon application artifact containing all the resources.

6. After step #5 is successful, traverse to **CarbonAppProject/target/** directory and find the generated carbon application with the name **ESBCAppProject_1.0.0.car.**
   This is the artifact containing all the resources (Proxies, Sequences, Endpoints, XSLT's) needed for this demo.

7. Start WSO2 ESB and deploy the above carbon application on to it. Once successfully deployed, you will see 4 different proxy services deployed (CommonProxy, FileBasedListerner, HTTPBasedListener and JMSBasedListener)

8. Also the two sequences mentioned above (EndpointSelectionSequence and CommonSequence) will also be deployed.

9. The registry resources (Endpoints and XSLT's) will also get added with this carbon application deployment. Along with this the recipient list xml resource will also be added as a registry resource. All of these resources can be found under the registry location conf:repository/demos/ where they have placed under different registry collections.

10. The demo uses a sample axis2 service comes with ESB as the HTTP backend. Follow the steps described in : [https://docs.wso2.com/display/ESB481/Setting+Up+the+ESB+Samples#SettingUptheESBSamples-Backend](https://docs.wso2.com/display/ESB481/Setting+Up+the+ESB+Samples#SettingUptheESBSamples-Backend) to deploy this service. Once this is successful, we can see that the service running at http://localhost:9000/services/SimpleStockQuoteService?wsdl

11. By default the recipient list xml (App1.xml) contains two endpoints (HttpEndpoint1 and JMSEndpoint1). So invoking any of the listeners with the correct requests will call both HttpEndpoint1 and JMSEndpoint1 via the CommonProxy.

12. Example requests messages for all three listener are found under resources/example-requests. An example curl command on HTTP/SOAP request for HTTPBasedListener is as follows.

    **curl -X POST http://localhost:8280/services/HTTPBasedListener.HTTPBasedListenerHttpSoap11Endpoint -H "ClientId : App1" -H "Content-Type: text/xml" -d @SOAP-Req.xml**

    You can see that we are sending a http header "ClientId" with the value "App1". This will basically select the App1.xml recipient list by the EndPointSelectionSequence.

13. Similarly an example JMS java client (QueueSender.java) is also available at the same location, which can be used with JMSBasedListerner. The example java client is given below. In here also, you can see that we are sending a JMS property "ClientId" with the value "App1".

```
        ConnectionFactory factory = new ActiveMQConnectionFactory("admin", "admin",
"tcp://localhost:61616");

        Connection connection = factory.createConnection();
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        Queue queue = session.createQueue("JMSIn");

        MessageProducer producer = session.createProducer(queue);

        TextMessage msg = session.createTextMessage();
        msg.setText(
            "<soapenv:Envelope
xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\">\n" +
            "      <soapenv:Body>" +
            "         <m0:CheckPriceRequest xmlns:m0=\"http://services.samples\">" +
            "            <m0:Code>WSO2</m0:Code>" +
            "         </m0:CheckPriceRequest>" +
            "      </soapenv:Body>\n" +
            "</soapenv:Envelope>");

        msg.setStringProperty("ClientId", "App1");
        producer.send(msg);
        session.close();
        connection.close();
```

The above client sends a message to a queue with the name "JMSIn" which will be picked up by the JMSBasedListener. The listener will process this message and get the JMS property with the name "ClientId" and set it as a mediation property and then hand over to the EndPointSelectionSequence.

14. For FileBasedListener, the file vfs-in.xml which is found under the example-requests can be used. In both JMS and File based approach, a SOAP request is sent as the content. Below is the content of the vfs-in.xml file. In here we are sending a custom soap header "ClientId" to trigger the recipient list selection.

```
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Header>
        <ns1:ClientId xmlns:ns1="http://custom.header.samples">App1</ns1:ClientId>
</soapenv:Header>
<soapenv:Body>
```

```
            <m0:CheckPriceRequest xmlns:m0="http://services.samples">
                  <m0:Code>WSO2</m0:Code>
            </m0:CheckPriceRequest>
      </soapenv:Body>
</soapenv:Envelope>
```

Note : You have to change the file locations on FileBasedListener proxy configuration with correct file path values.

When this file is placed in the location listened by the FileBasedListener, it will be picked up and processed. While processing, it will get the “ClientId” soap header and add it as a mediation property and then hand over to the EndPointSelectionSequence.

15. We can see that all three listener’s follow a similar pattern by setting a mediation property from the incoming message (Http Header, JMS Property, Soap Header) and the hand it over to EndPointSelectionSequence.