WSO2 Enterprise Integrator 6.5.0

Labkit

Developer Advanced - ESB Profile

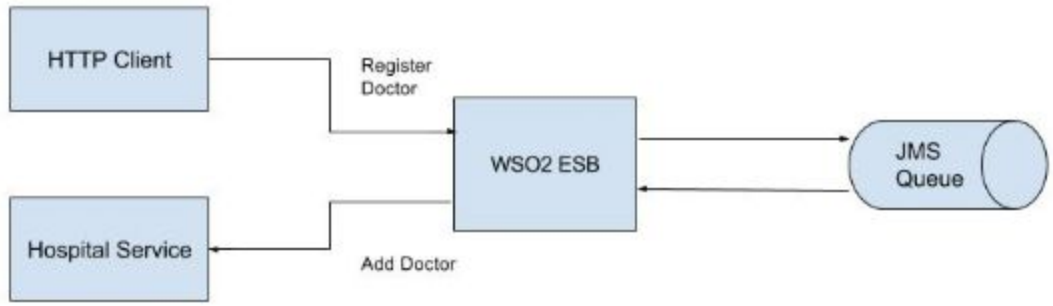
training@wso2.com

Table of Contents

Lab: WSO2 Enterprise Integrator as a JMS Producer and Consumer

Training Objective

Configure WSO2 Enterprise Integrator(WSO2 EI) to listen to an ActiveMQ queue. This introduces students to using WSO2 EI as a JMS producer and a consumer to communicate with a sample Health Care service as illustrated in the diagram below.



With the JMS transport, you can decouple the message sender and receiver. This means that the message sender can send messages reliably regardless of the fact whether the system that receives the messages is running or not during that particular time. In this scenario, an HTTP client is used, to send a request to register a new doctor to the sample Healthcare service through WSO2 EI, which is configured as the JMS producer.

Once messages are queued on the JMS Queue within the ActiveMQ broker, WSO2 EI, which is configured as a JMS consumer can connect to the queue and asynchronously consume the message/s.

High Level Steps

- Configure Apache ActiveMQ
- Configure JMS transport for ActiveMQ in WSO2 EI
- Create artifacts in WSO2 EI
- Test WSO2 EI as the JMS Producer and Consumer

Detailed Instructions

Configure Apache ActiveMQ

1. Download and install Apache ActiveMQ.
2. Start the ActiveMQ Broker by executing the following command:

```
./activemq console
```

(If you are using Mac OS, execute the following command: `sh activemq console`)

Configure the JMS transport with ActiveMQ

1. Copy the following client libraries from the `<ACTIVEMQ_HOME>/lib/` directory to the `<EI_HOME>/lib/` directory.

   For ActiveMQ 5.8.0 and above

   `activemq-broker-5.8.0.jar`
   `activemq-client-5.8.0.jar`
   `activemq-kahadb-store-5.8.0.jar`
   `geronimo-jms_1.1_spec-1.1.1.jar`
   `geronimo-j2ee-management_1.1_spec-1.0.1.jar`
   `geronimo-jta_1.0.1B_spec-1.0.1.jar`
   `hawtbuf-1.9.jar`
   `Slf4j-api-1.6.6.jar`
   `activeio-core-3.1.4.jar` (available in the `<ACTIVEMQ_HOME>/lib/optional/` directory)

   For earlier versions of ActiveMQ

   `activemq-core-5.5.1.jar`
   `geronimo-j2ee-management_1.0_spec-1.0.jar`
   `geronimo-jms_1.1_spec-1.1.1.jar`

2. To configure the JMS transport listener to listen to a JMS queue, uncomment the following listener configuration related to ActiveMQ in the `<EI_HOME>/conf/axis2/axis2.xml` file.

```
<!--Uncomment this and configure as appropriate for JMS transport
support, after setting up your JMS environment (e.g. ActiveMQ)-->
<transportReceiver name="jms"
class="org.apache.axis2.transport.jms.JMSListener">
       <parameter name="myTopicConnectionFactory" locked="false">
           <parameter name="java.naming.factory.initial"
locked="false">org.apache.activemq.jndi.ActiveMQInitialContextFactory</p
arameter>
           <parameter name="java.naming.provider.url"
locked="false">tcp://localhost:61616</parameter>
           <parameter name="transport.jms.ConnectionFactoryJNDIName"
locked="false">TopicConnectionFactory</parameter>
            <parameter name="transport.jms.ConnectionFactoryType"
locked="false">topic</parameter>
       </parameter>
```

```
        <parameter name="myQueueConnectionFactory" locked="false">
            <parameter name="java.naming.factory.initial"
locked="false">org.apache.activemq.jndi.ActiveMQInitialContextFactory</p
arameter>
            <parameter name="java.naming.provider.url"
locked="false">tcp://localhost:61616</parameter>
            <parameter name="transport.jms.ConnectionFactoryJNDIName"
locked="false">QueueConnectionFactory</parameter>
             <parameter name="transport.jms.ConnectionFactoryType"
locked="false">queue</parameter>
        </parameter>

        <parameter name="default" locked="false">
            <parameter name="java.naming.factory.initial"
locked="false">org.apache.activemq.jndi.ActiveMQInitialContextFactory</p
arameter>
            <parameter name="java.naming.provider.url"
locked="false">tcp://localhost:61616</parameter>
            <parameter name="transport.jms.ConnectionFactoryJNDIName"
locked="false">QueueConnectionFactory</parameter>
             <parameter name="transport.jms.ConnectionFactoryType"
locked="false">queue</parameter>
        </parameter>
    </transportReceiver>
```

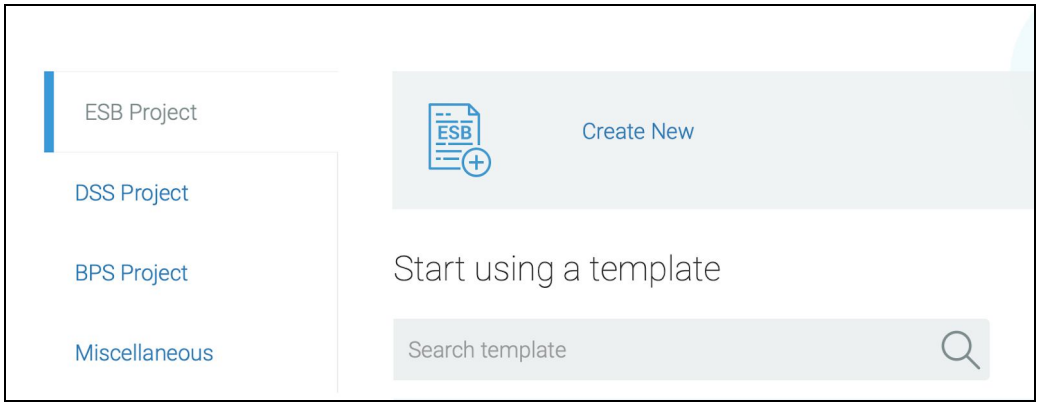
3. To configure a JMS transport sender to send messages to a JMS queue, uncomment the following configuration in the `<EI_HOME>/conf/axis2/axis2.xml` file.

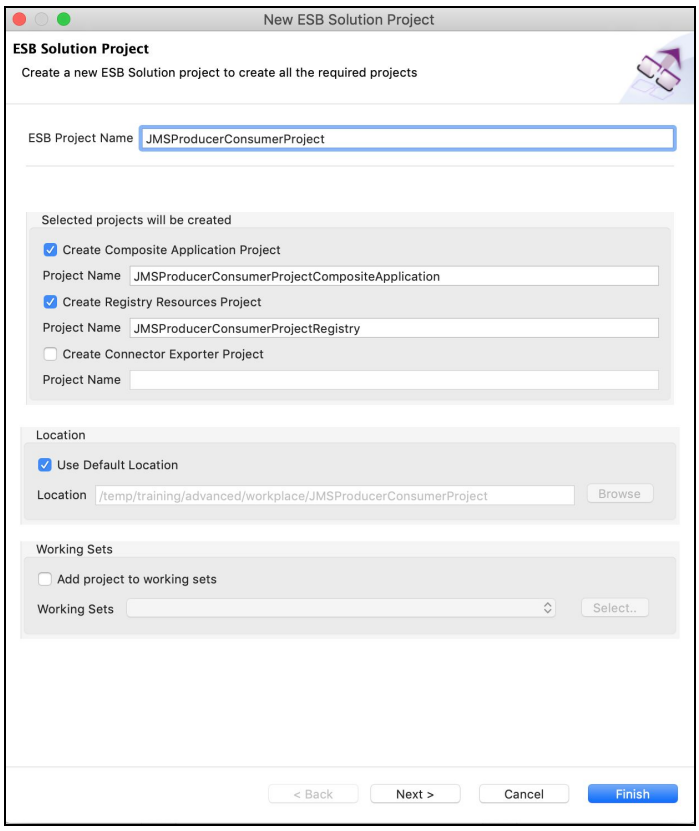
```
<transportSender name="jms"
class="org.apache.axis2.transport.jms.JMSSender"/>
```

Create artifacts in WSO2 EI

Create the Project

1. Open the **Developer Studio Dashboard** and create an **ESB Config Project** named `JMSProducerConsumerProject`.

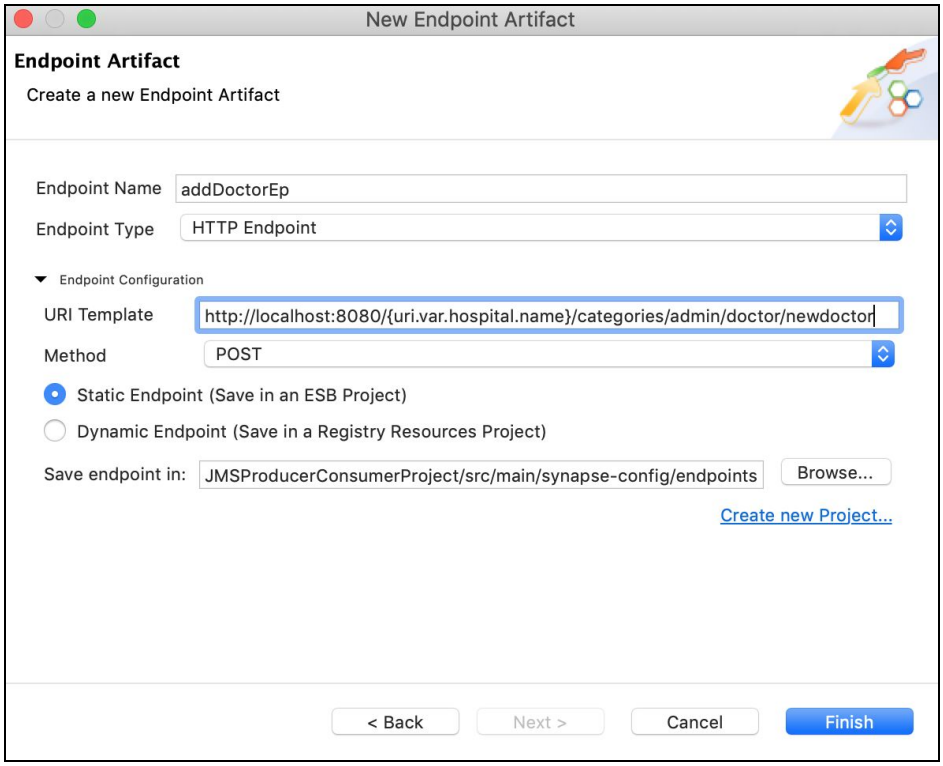




Create the Endpoint

1. Right click on the project and select **New** -> **Endpoint** to create a new HTTP Endpoint named `addDoctorEp` to send the doctor details to the backend service.

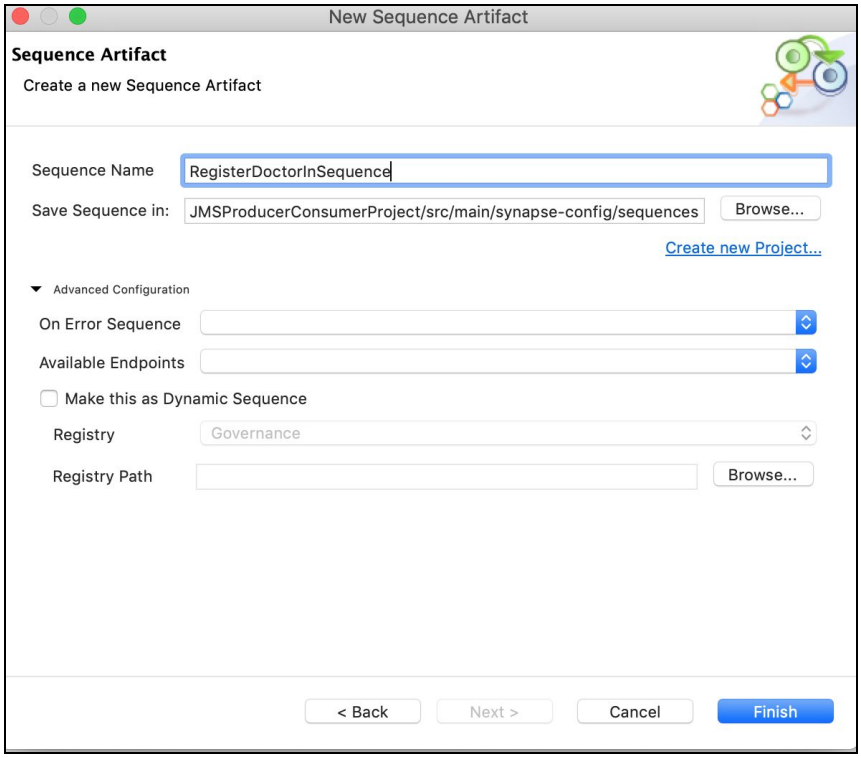
Field	Value
Endpoint Name	addDoctorEp
Endpoint Type	HTTP Endpoint
URI Template	http://localhost:8080/{uri.var.hospital.name}/categories/admin/doctor/newdoctor
Method	POST



Create the Sequences

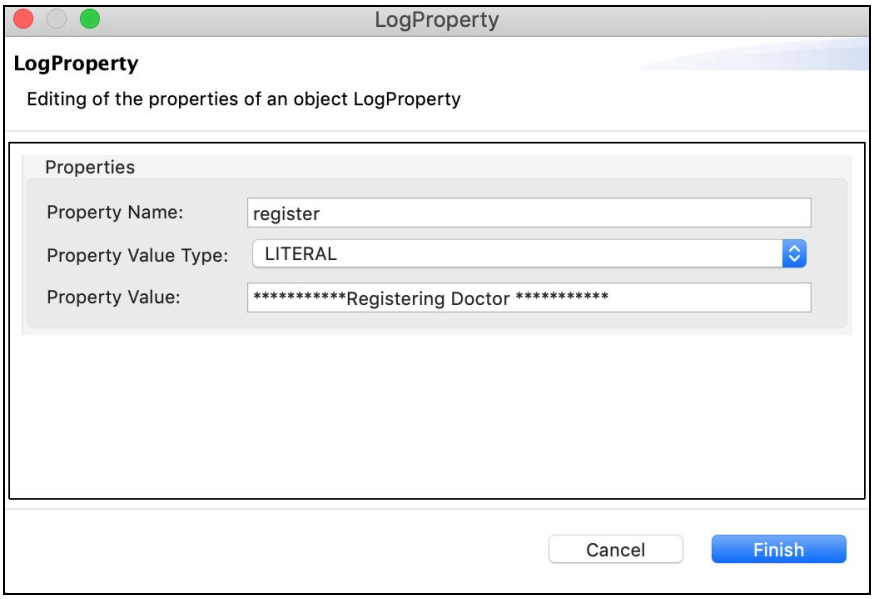
Create the `RegisterDoctorInSequence` Sequence

1. Right click on the project and select **New** -> **Sequence** to create a new sequence named `RegisterDoctorInSequence`. This will send the doctor details to the JMS queue.



2. Drag and drop a Log mediator to the `RegisterDoctorInSequence` sequence and change its properties as follows.

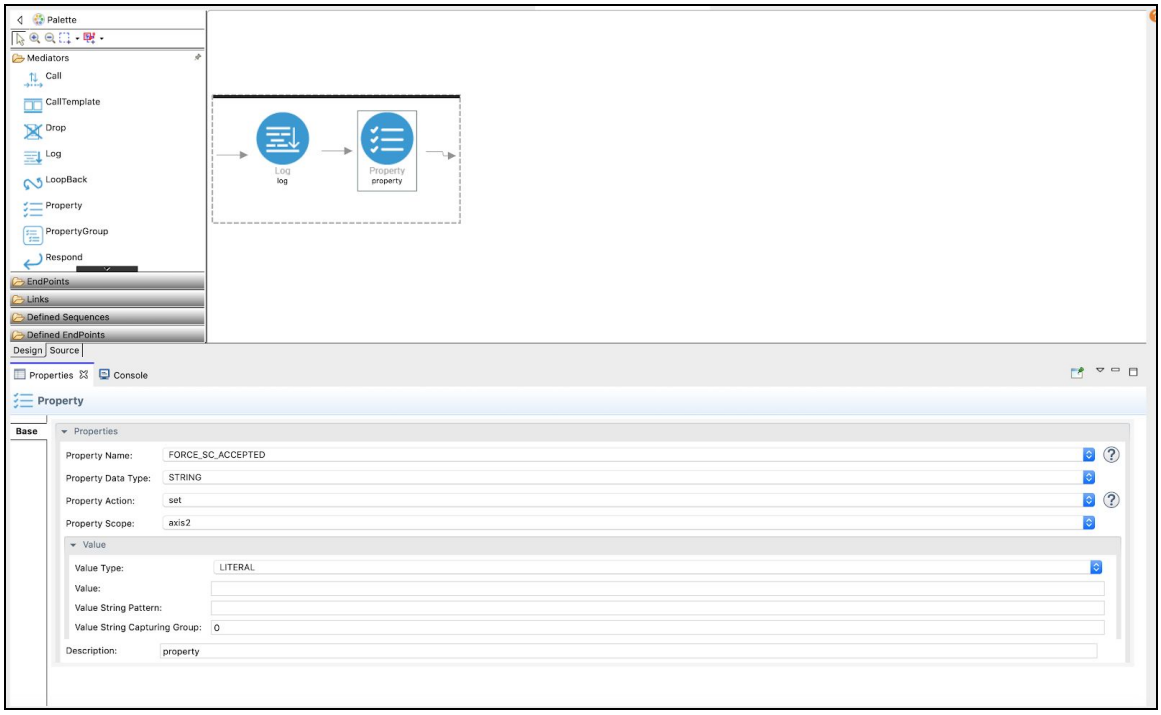
Property	Value
Log Category	INFO
Log Level	FULL
Log Separator	,
Properties	Double-click on the value field and add a new property as follows:



Name: register
Value/Expression: `**********Registering Doctor **********`

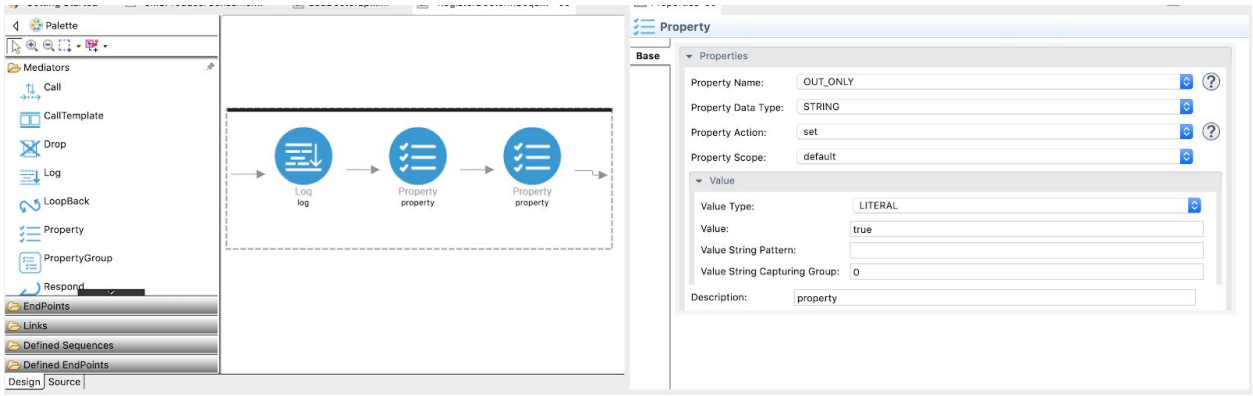
3. Drag and drop a Property mediator to the `RegisterDoctorInSequence` sequence and change its properties as follows.

Property	Value
Property Name	New Property
New Property Name	FORCE_SC_ACCEPTED
Value	true
Property Scope	axis2

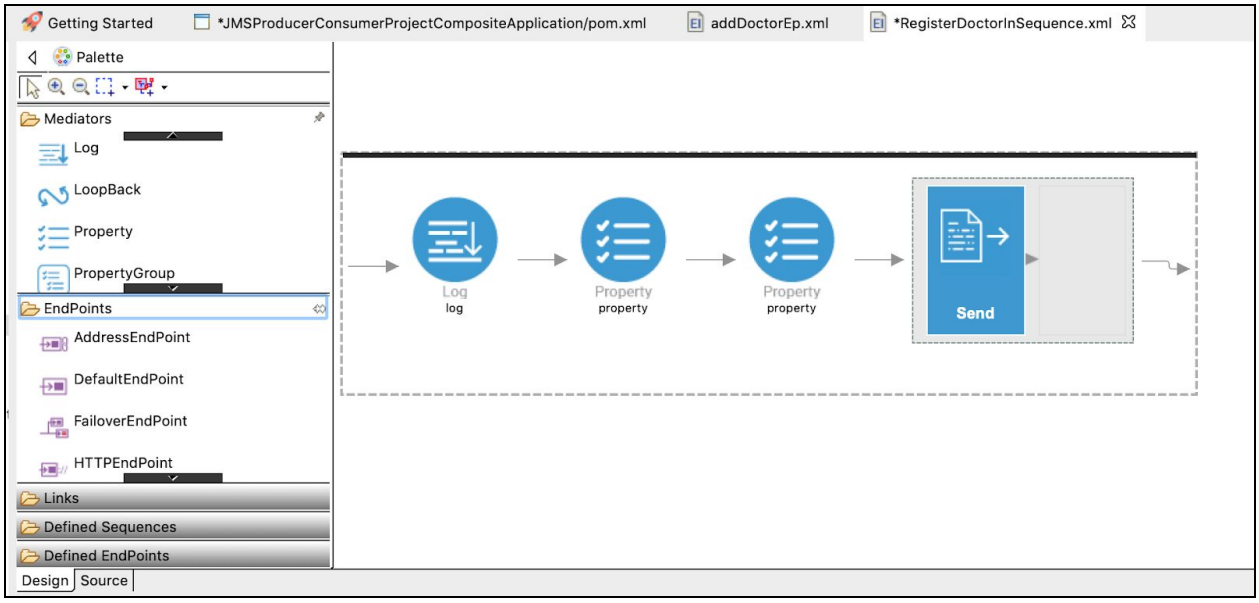


4. Insert another Property mediator to the `RegisterDoctorInSequence` sequence and change its properties as follows.

Property	Value
Property Name	New Property
New Property	OUT_ONLY
Value	true



5. Insert a Send mediator to the `RegisterDoctorInSequence` sequence.



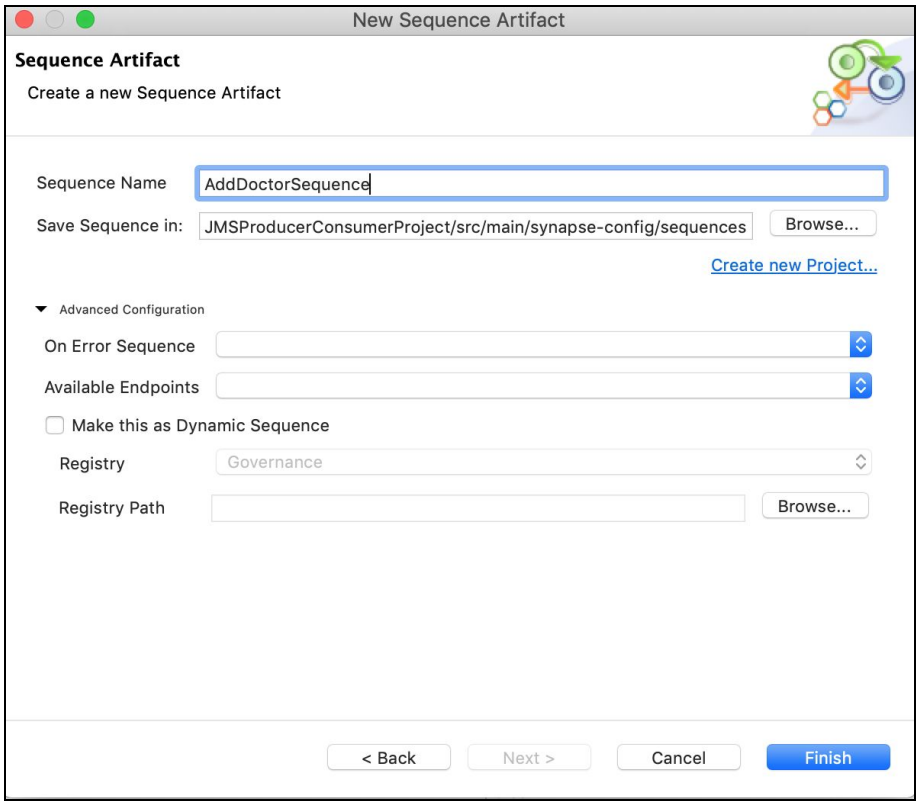
6. Insert an Address Endpoint to the adjoining cell of the Send mediator in the `RegisterDoctorInSequence` sequence and change its properties as follows.

Property	Value
URI	jms:/RegisterDoctorService?transport.jms.ConnectionFactoryJNDIName=QueueConnectionFactory&java.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory&java.naming.provider.url=tcp://localhost:61616&transport.jms.DestinationType=queue

***NOTE**: EI Tooling will escape & > &*

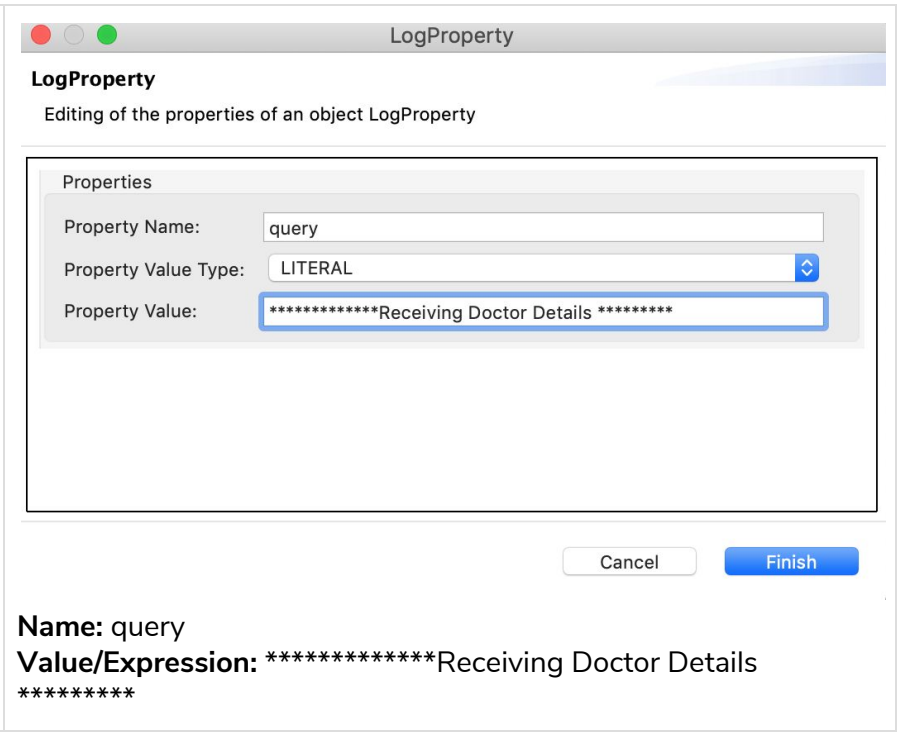
Create the `AddDoctorSequence` Sequence

7. Right click on the project and select **New** -> **Sequence** to create a new sequence named `AddDoctorSequence`. This will process the doctor details received from the queue before sending to the backend service.

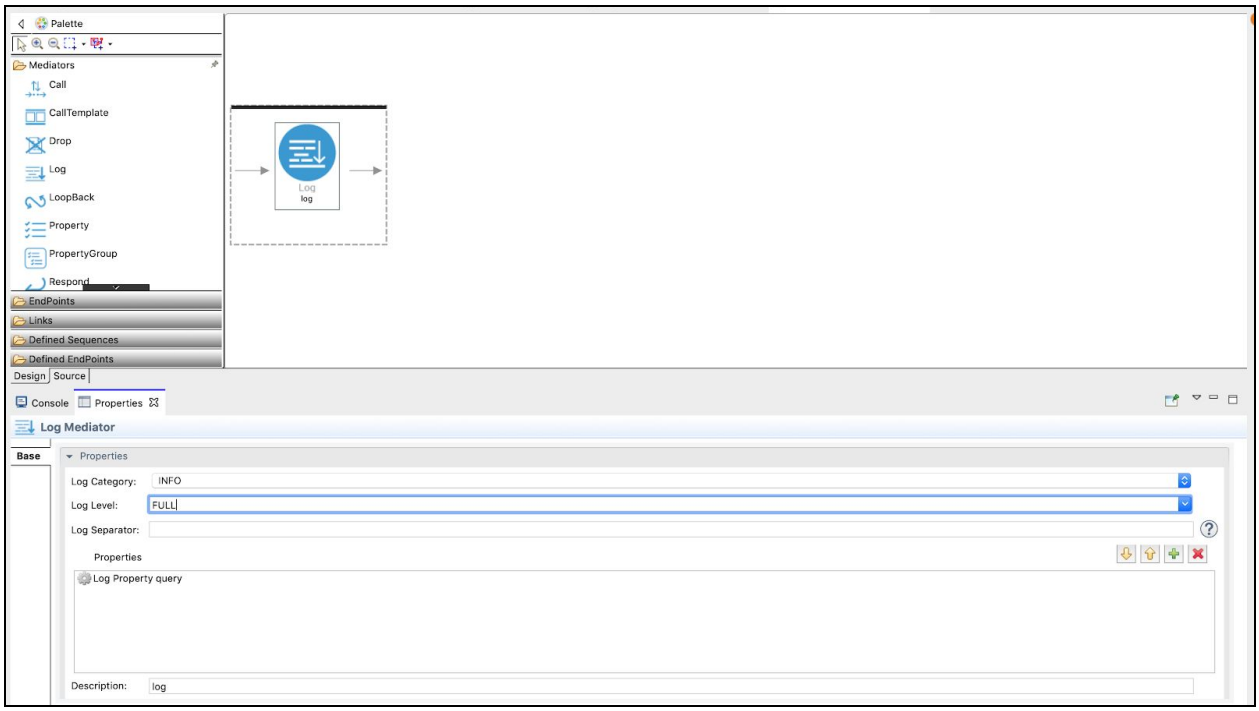


8. Drag and drop a Log mediator to the `AddDoctorSequence` sequence and change its properties as follows.

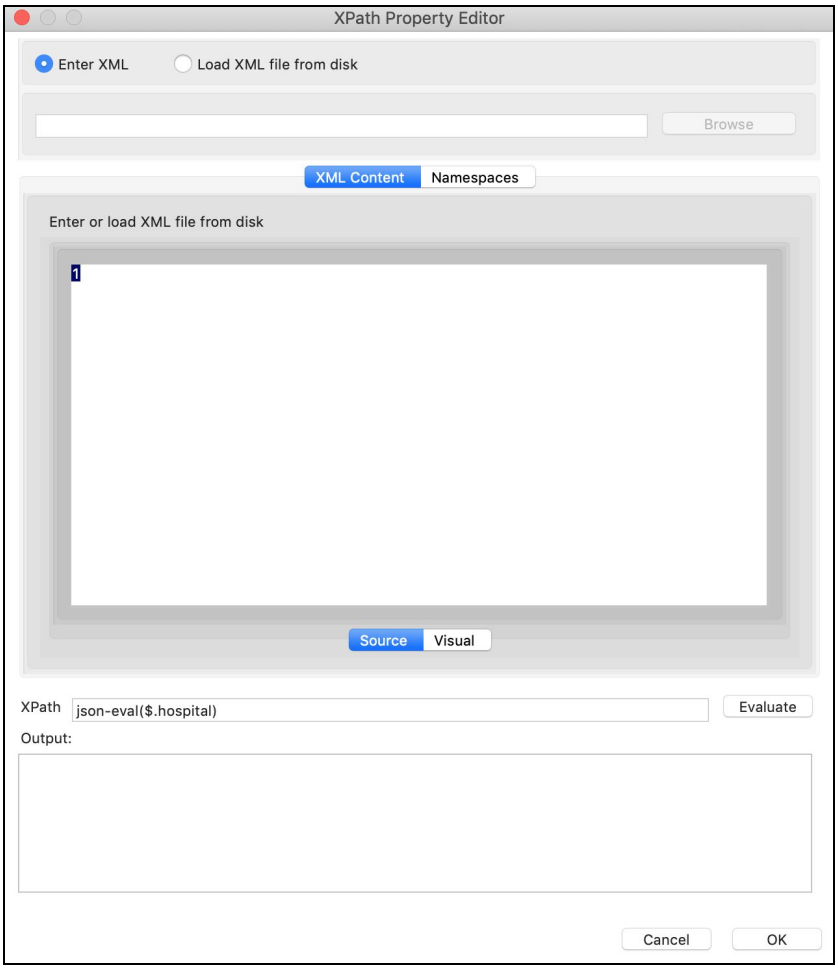
Property	Value
Log Category	INFO
Log Level	FULL
Log Separator	,
Properties	Double-click on the value field and add a new property as follows:

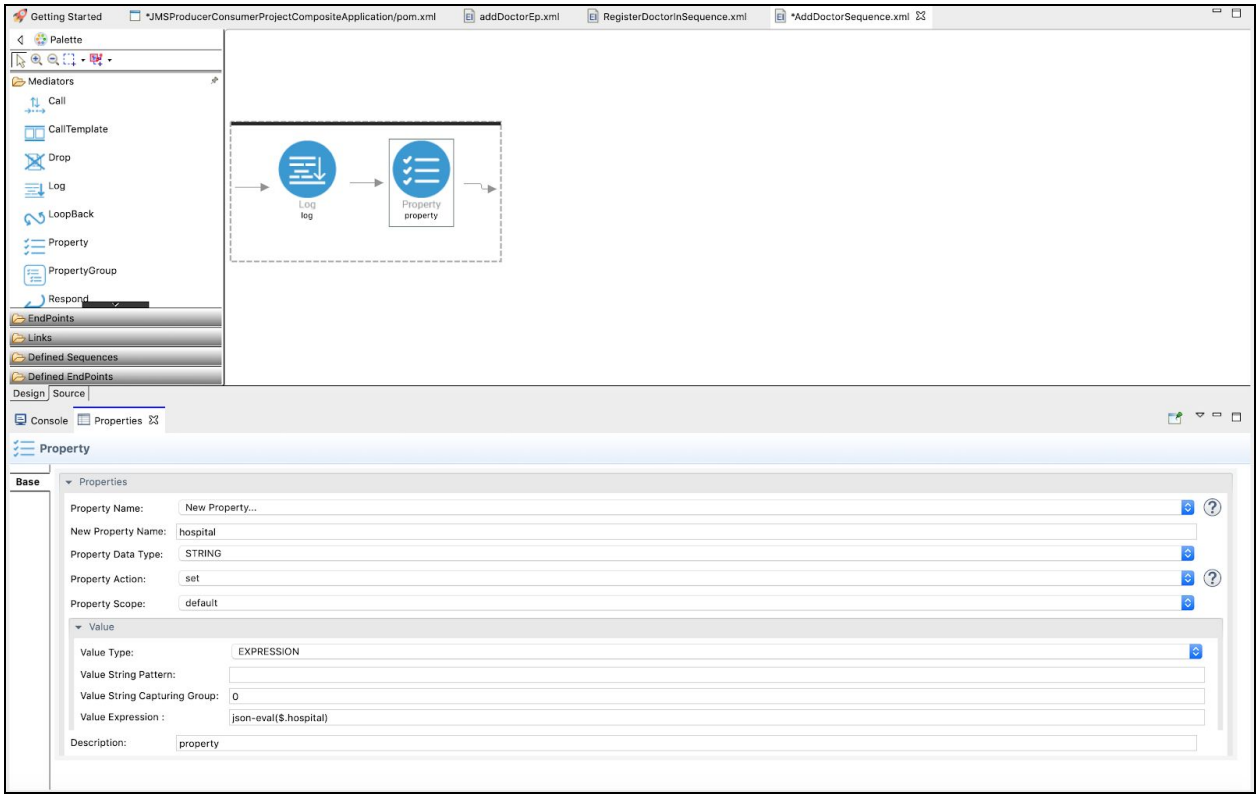


Name: query
Value/Expression: **********Receiving Doctor Details *********



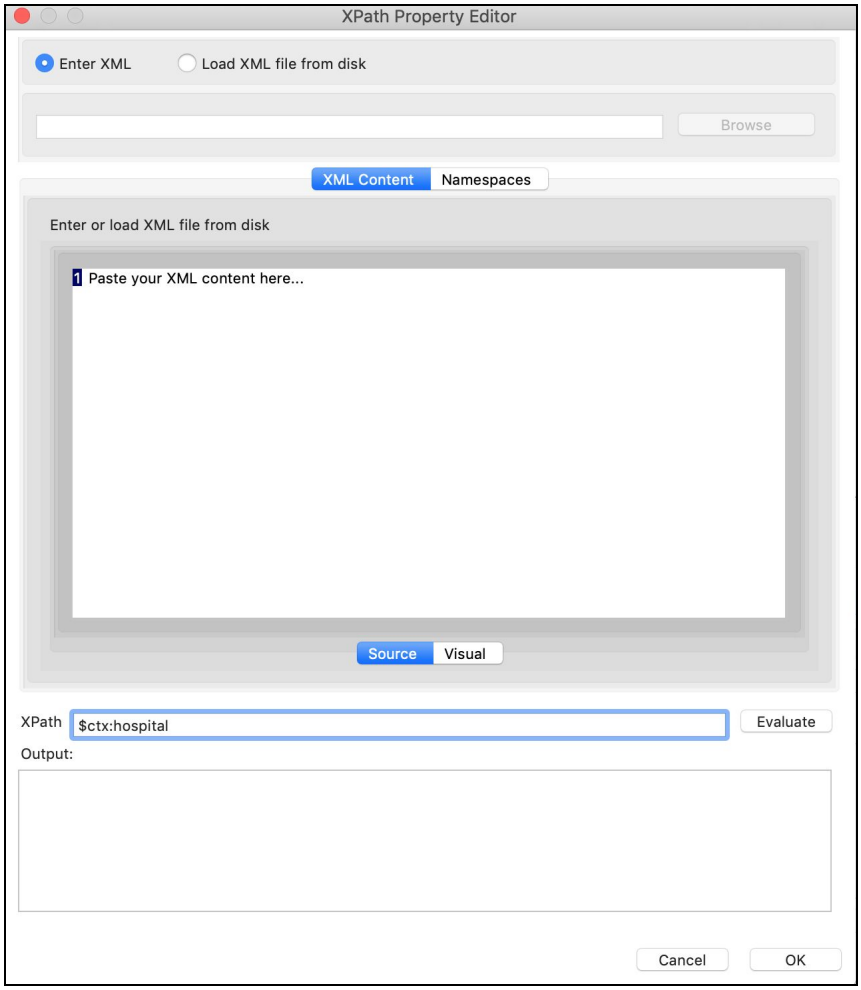
9. Drag and drop a Property mediator to the `AddDoctorSequence` sequence and change its properties as follows.

Property	Value
Property Name	New Property
New Property Name	hospital
Value Type	EXPRESSION
Value Expression	Click on the value field and enter the `json-eval($.hospital)` expression as follows: 

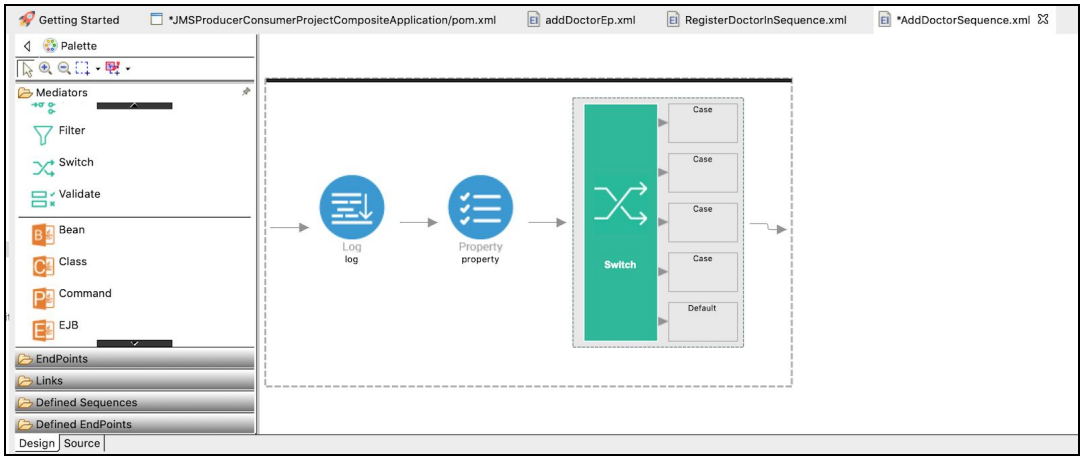


10. Drag and drop a Switch mediator to the `AddDoctorSequence` sequence.

11. In the **Properties** tab of the Switch mediator click on the value field of the **Source Xpath** and enter the following Xpath value: `$ctx:hospital`



12. Right-click on the Switch mediator and click **Add/Remove Case** and type 4 for the **Number of branches** to add four cases.



13. Double click on one Case and add the following values as **Case Branches**.
    - Case 1 - grand oak community hospital
    - Case 2 - clemency medical center
    - Case 3 - pine valley community hospital
    - Case 4 - willow gardens general hospital

14. Drag and drop a Property mediator to the first case and change its properties as follows.

Property	Value
Property Name	New Property
New Property Name	uri.var.hospital.name
Value	grandoaks

15. Add Property mediators to the other three cases and change the properties as follows.

Case 2:

Property	Value
Property Name	New Property
New Property Name	uri.var.hospital.name
Value	clemency

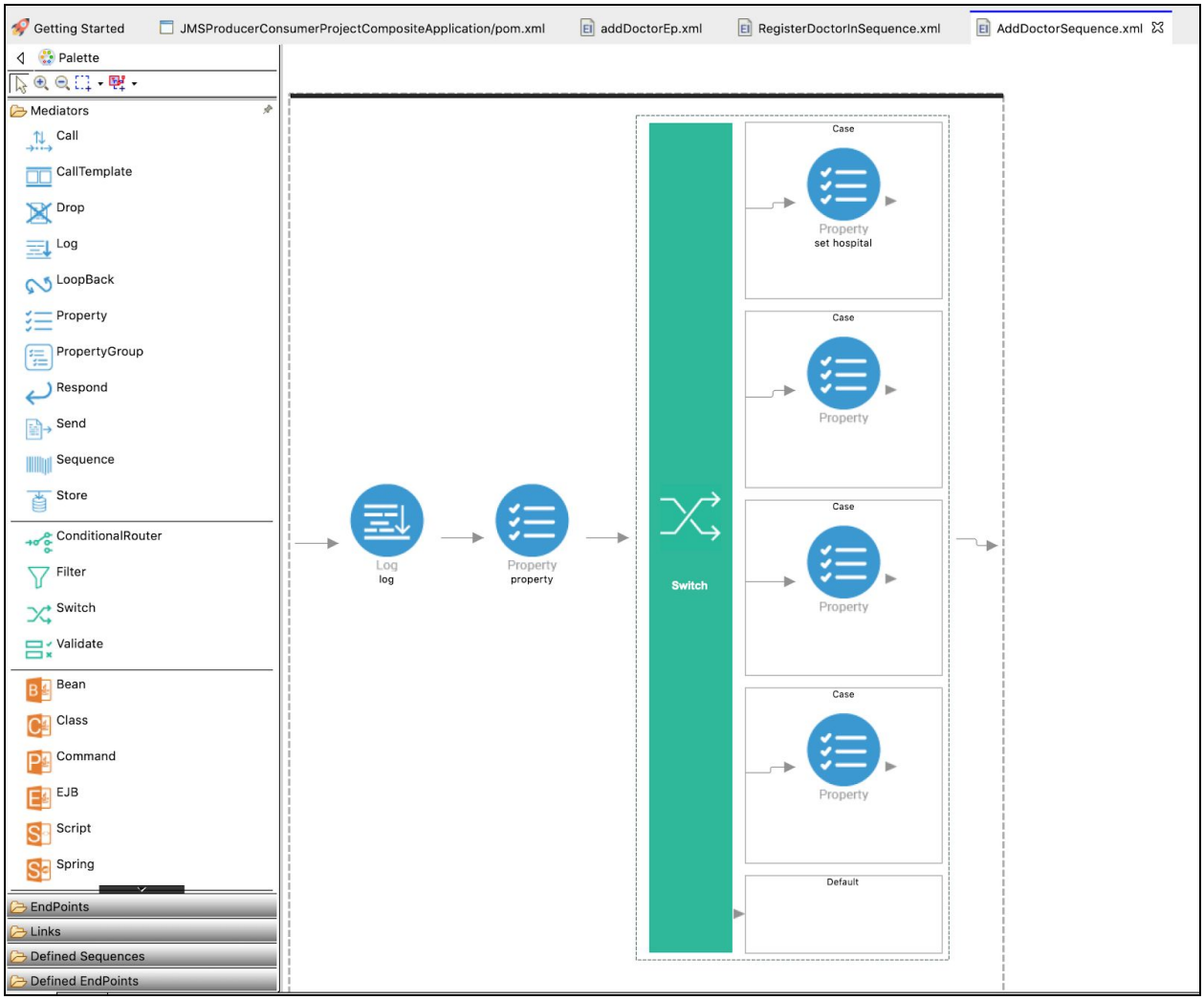
Case 3:

Property	Value

Property Name	New Property
New Property Name	uri.var.hospital.name
Value	pinevalley

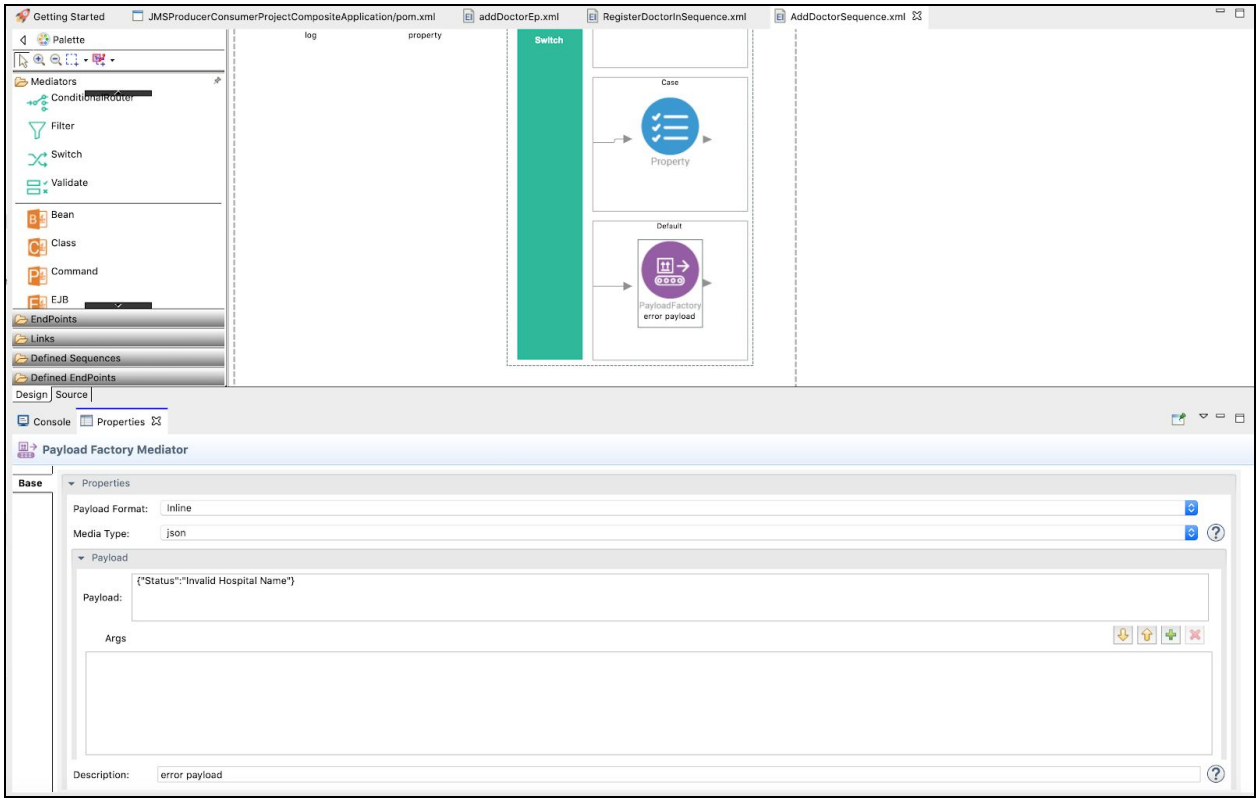
Case 4:

Property	**Value**
Property Name	New Property
New Property Name	uri.var.hospital.name
Value	willowogardens

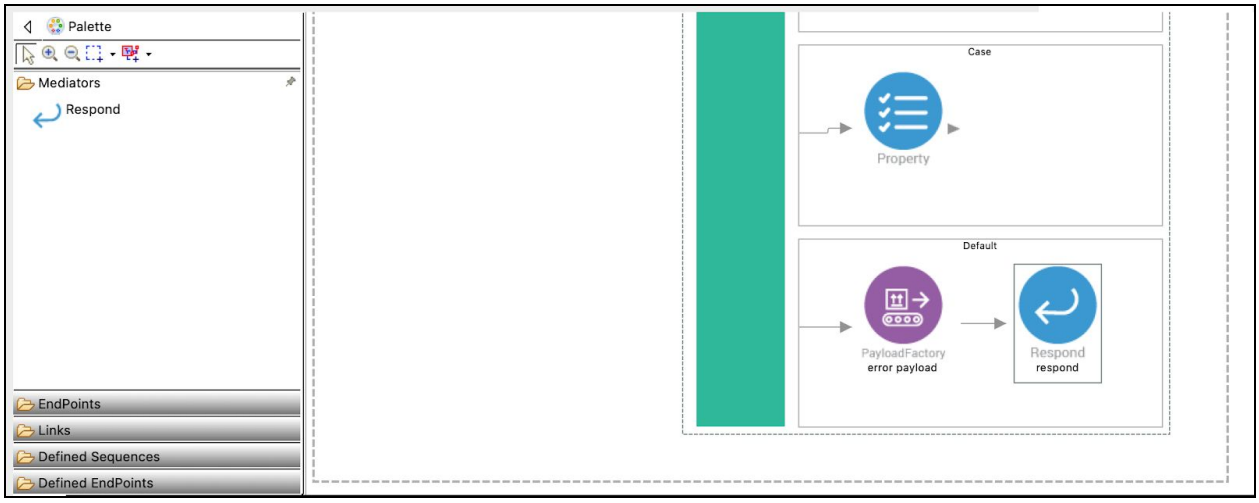


16. Drag and drop a Payload Factory mediator to the **Default** cell of the Switch mediator and change its properties as follows.

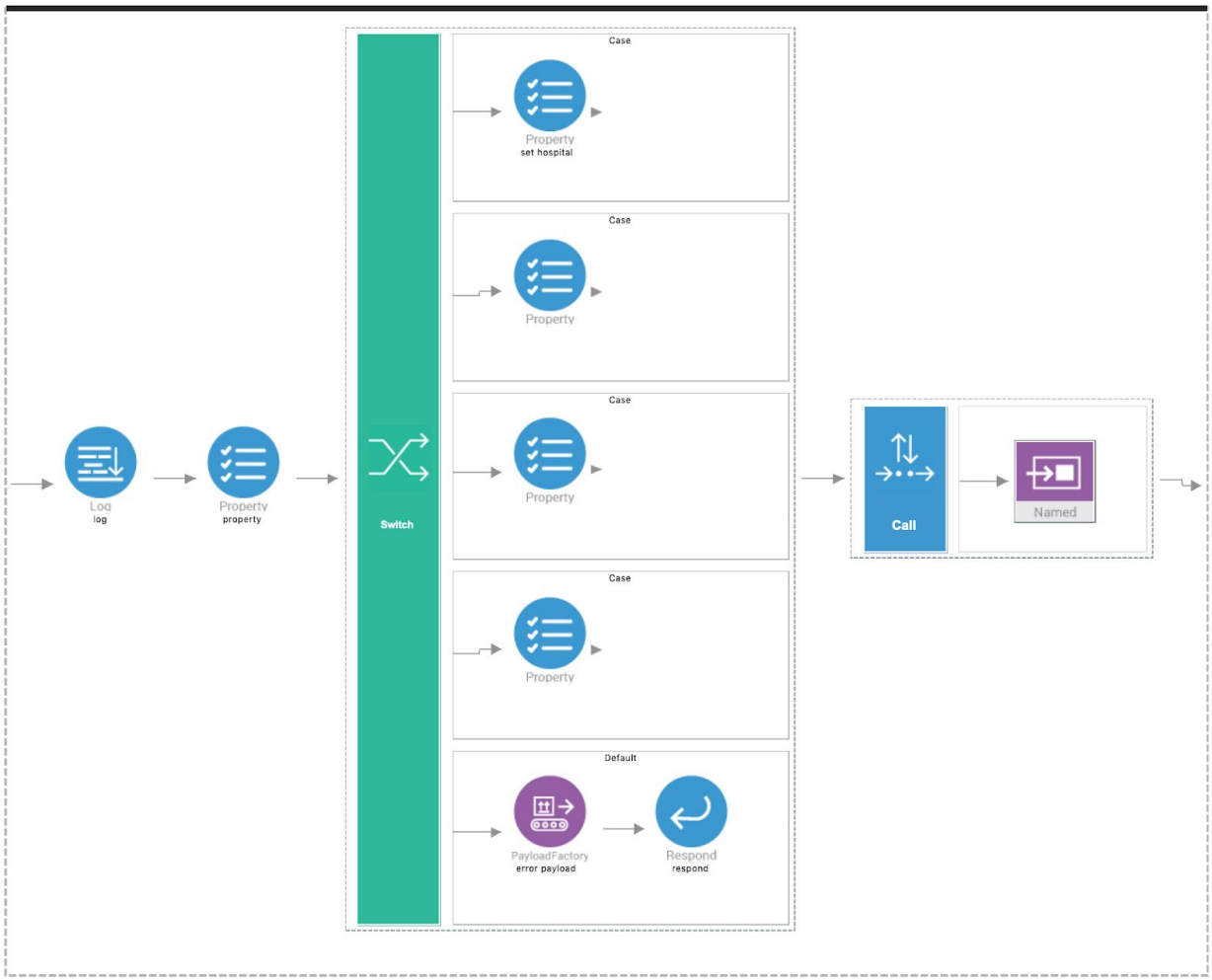
Property	Value
Payload Format	Inline
Payload	{"Status":"Invalid Hospital Name"}
Media Type	json



17. Drag and drop a Respond mediator after the PayloadFactory mediator to the **Default** cell of the Switch mediator.

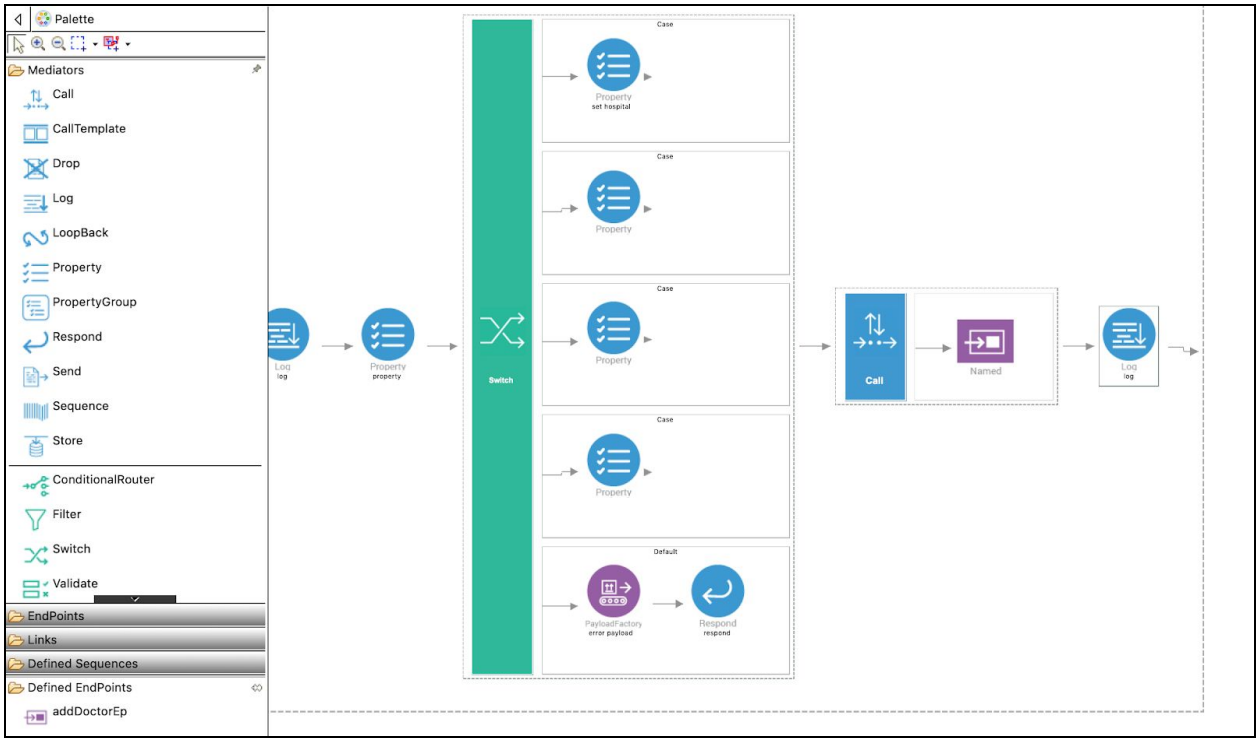


18. Drag and drop a Call mediator to the `addDoctorSequence` sequence and drag and drop the `addDoctorEp` endpoint from the **Defined Endpoints** panel of the **Palette** to the empty cell adjoining the Call mediator.



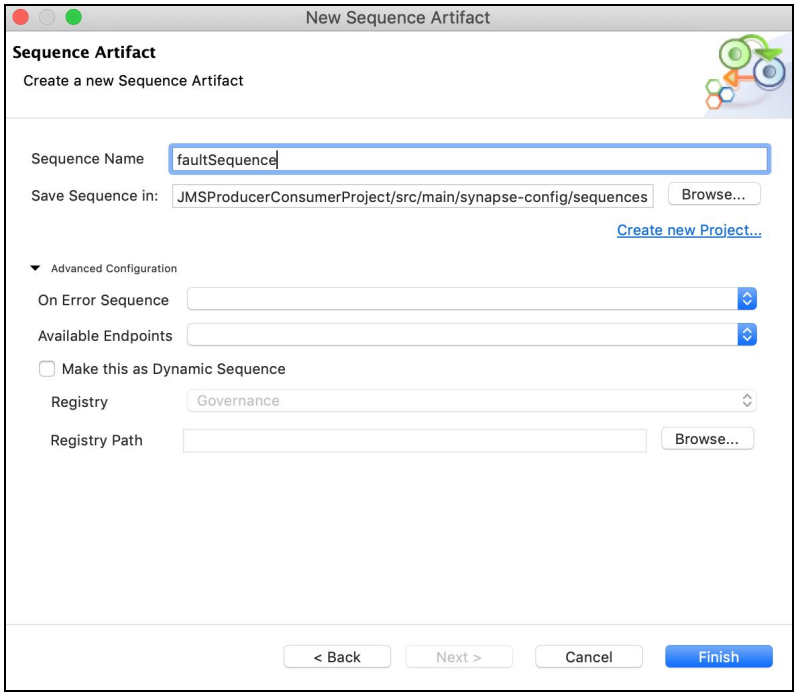
19. Drag and drop a Log mediator to the `addDoctorSequence` sequence and change its properties as follows.

Property	Value
Log Category	INFO
Log Level	FULL



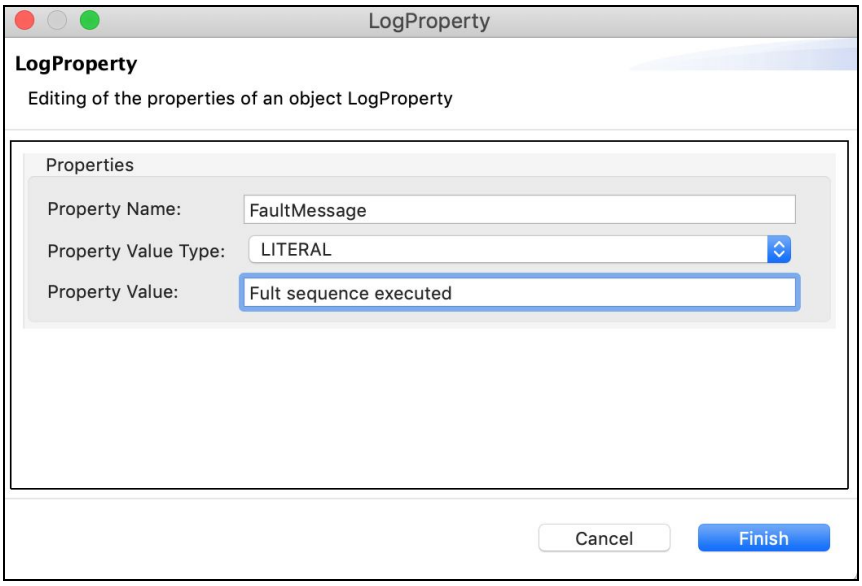
Create the `FaultSequence` Sequence

20. Right click on the project and select **New** -> **Sequence** to create a new sequence named `faultSequence`. If an error occurs, doctor details are sent to this sequence.



21. Drag and drop a Log mediator to the `faultSequence` sequence and change the following properties.

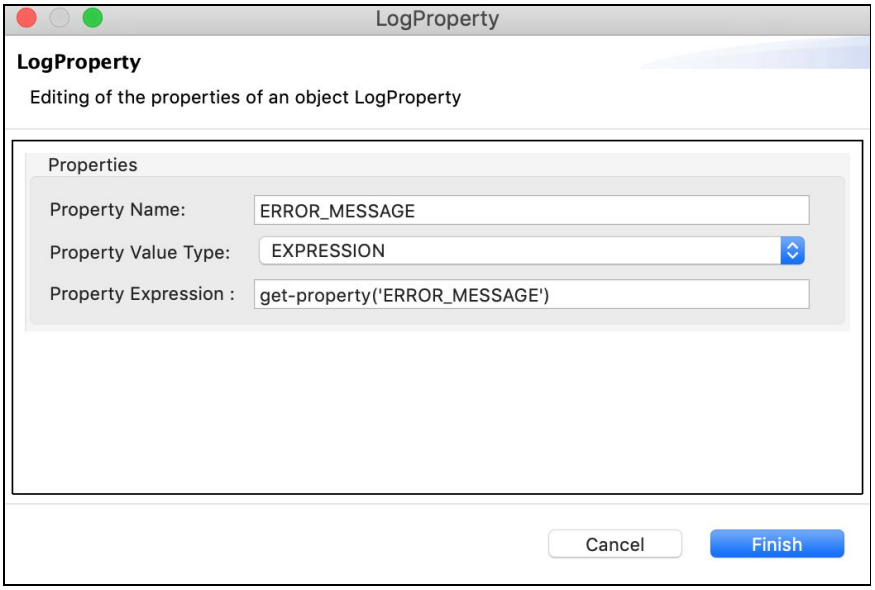
Property	Value
Log Category	INFO
Log Level	CUSTOM
Properties	Click on the value field, click **New**, and enter the following property values: **Name:** FaultMessage **Value/Expression:** Fault Sequence Executed



Name: ERROR_MESSAGE
Type: EXPRESSION

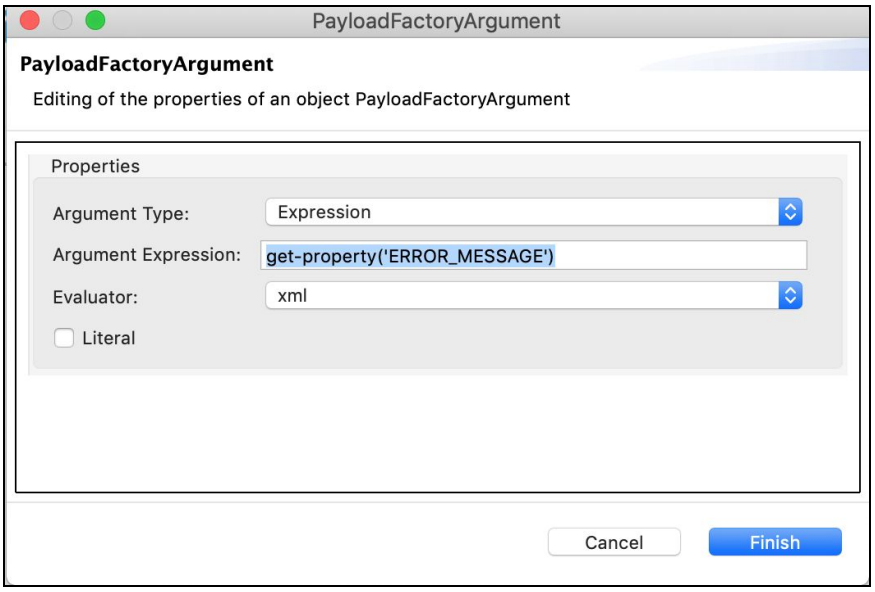
Value/Expression: Click on the value field and enter the expression and the namespace as shown below.

- **Expression:** get-property('ERROR_MESSAGE')
- **Prefix:** ns
- **URI:** http://org.apache.synapse/xsd

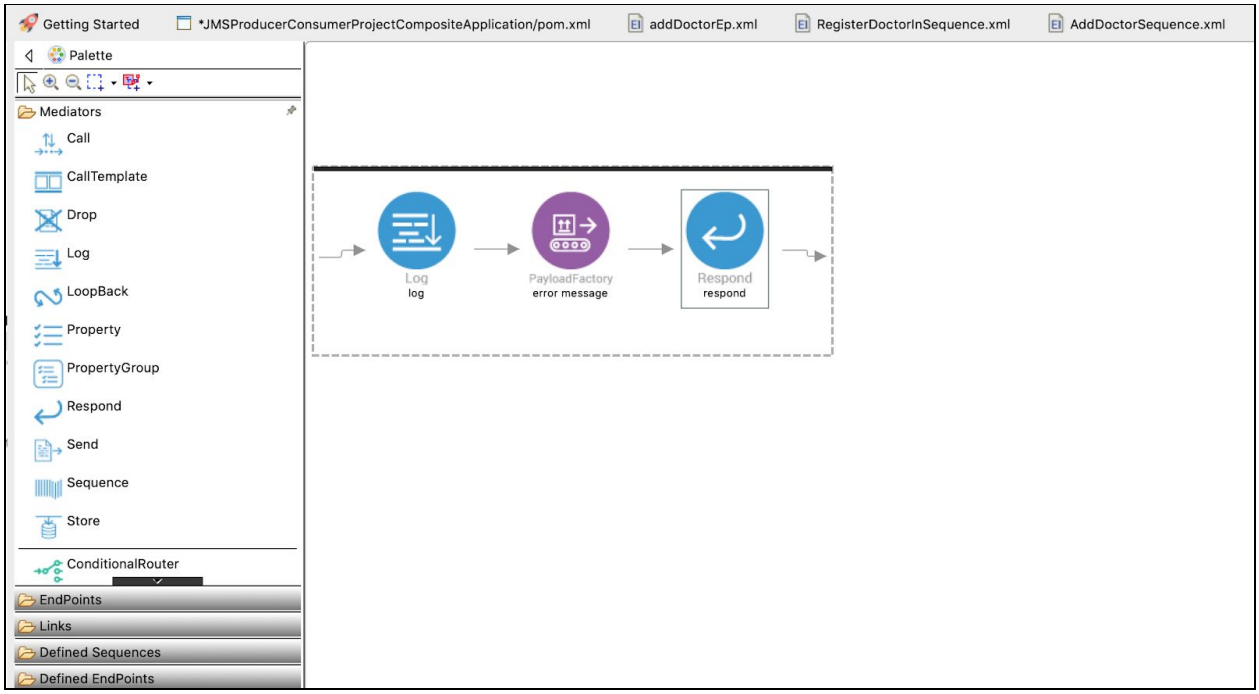


Click **Add** and click **OK**.

22. Drag and drop a Payload Factory mediator to the `faultSequence` sequence and change the following properties.

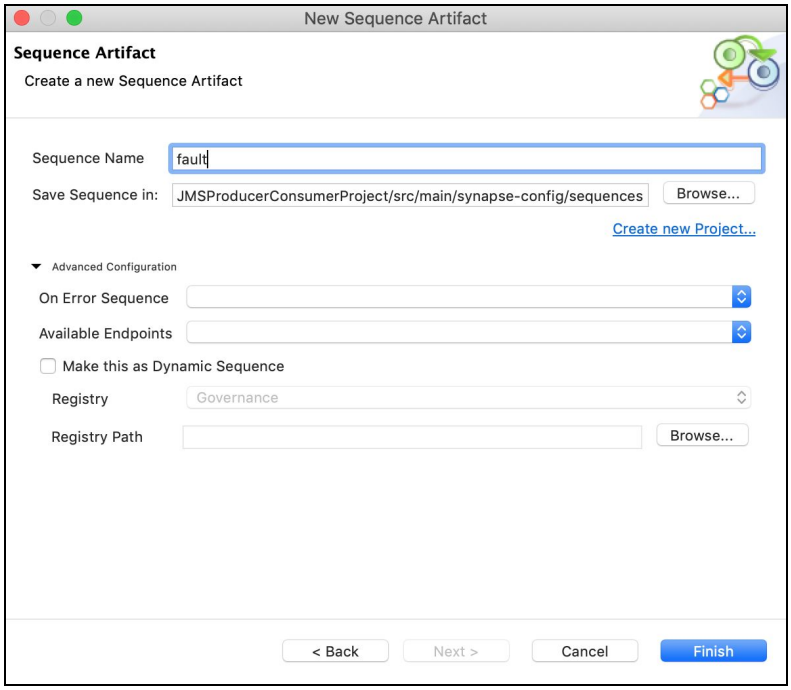
Property	Value
Media Type	json
Payload Format	Inline
Payload	Click on the value field and enter following: `{"Status":"Error Occurred While Processing the Request", "Message":"$1"}`
Args	Click on the value field and enter the following: ● **Type:** Expression ● **Value:** get-property('ERROR_MESSAGE') ● **Evaluator:** xml 

23. Drag and drop a Respond mediator to the `faultSequence` sequence.



Create the `Fault` Sequence

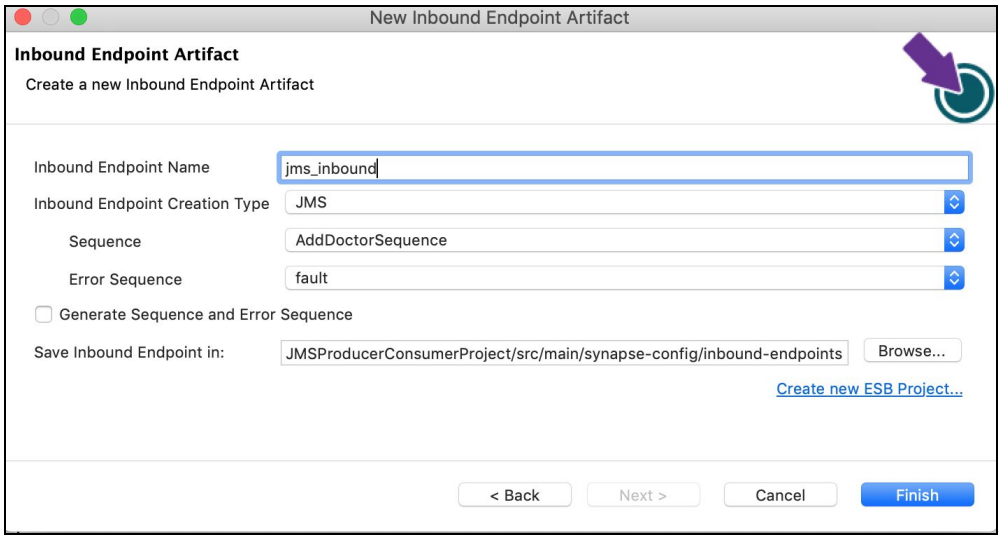
24. Right click on the project and select **New** -> **Sequence** to create a new Sequence named `fault`.



Create the Inbound Endpoint

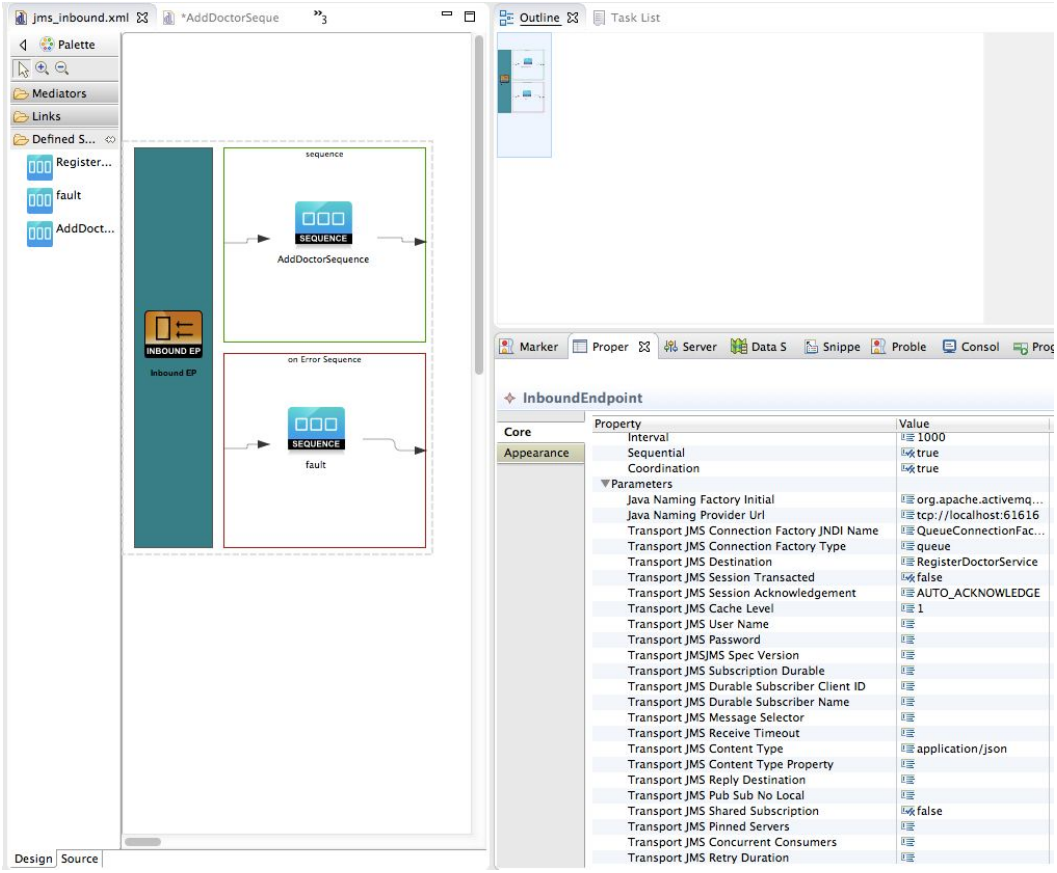
1. Right click on the project and select **New** -> **Inbound Endpoint** to create a new inbound endpoint named `jms_inbound`. (Select `JMS` as the **Inbound Endpoint Creation Type**). This will check the JMS queue periodically for messages. When a message is available, it will inject that message to a specified sequence.

   Select `AddDoctorSequence` as the sequence and `Fault` as the Error Sequence.



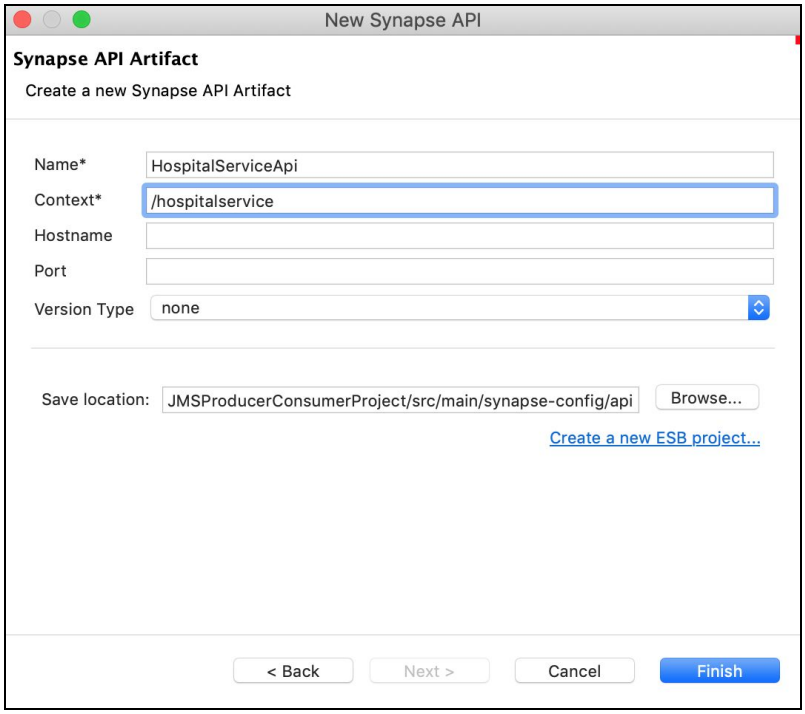
2. Double-click on the inbound endpoint and change the following properties.

Property	Value
Interval	1000
Sequential	true
Coordination	true
Java Naming Factory Initial	org.apache.activemq.jndi.ActiveMQInitialContextFactory
Java Naming Provider URL	tcp://localhost:61616
Transport JMS Connection Factory JNDI Name	QueueConnectionFactory
Transport JMS Connection Factory Type	queue
Transport JMS Destination	RegisterDoctorService
Transport JMS Session Transacted	false
Transport JMS Session Acknowledgement	AUTO_ACKNOWLEDGE
Transport JMS Cache Level	1
Transport JMS Content Type	application/json
Transport JMS Shared Subscription	false



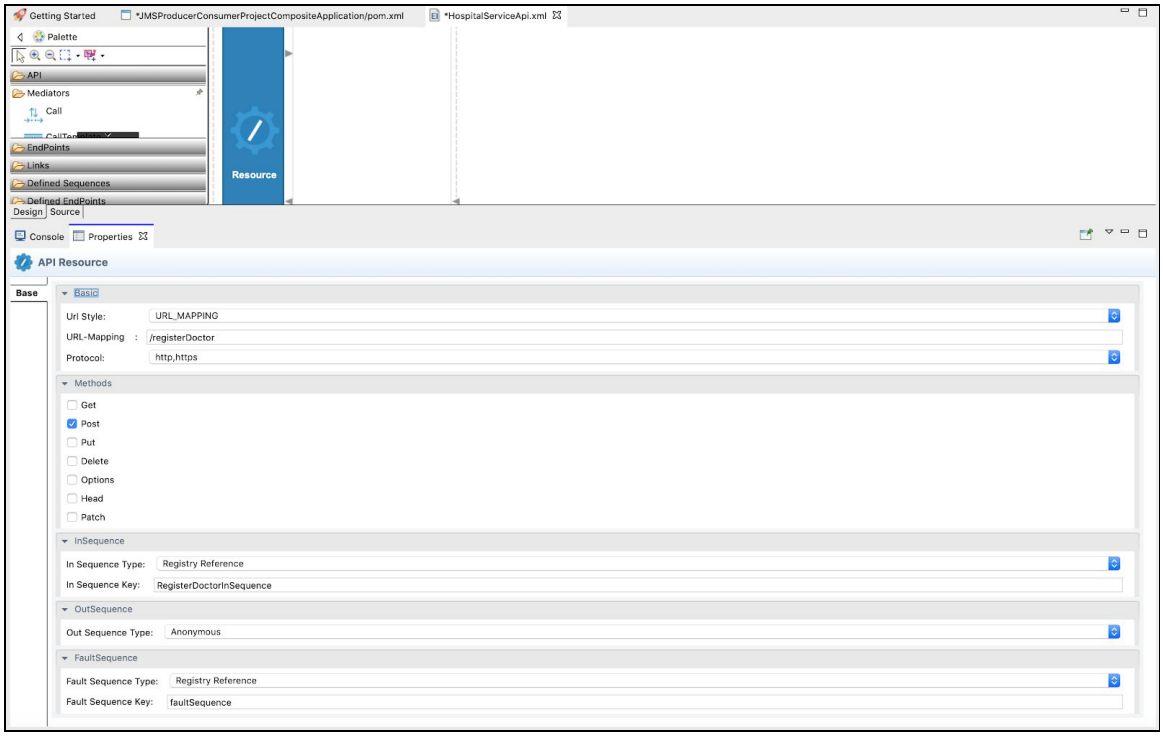
Create the API

1. Right click on the project and select **New** -> **REST API** to create a new API named `HospitalServiceApi`. (Give the Context as `/hospitalservice`.)



2. Change the following properties of the `HospitalServiceApi` API.

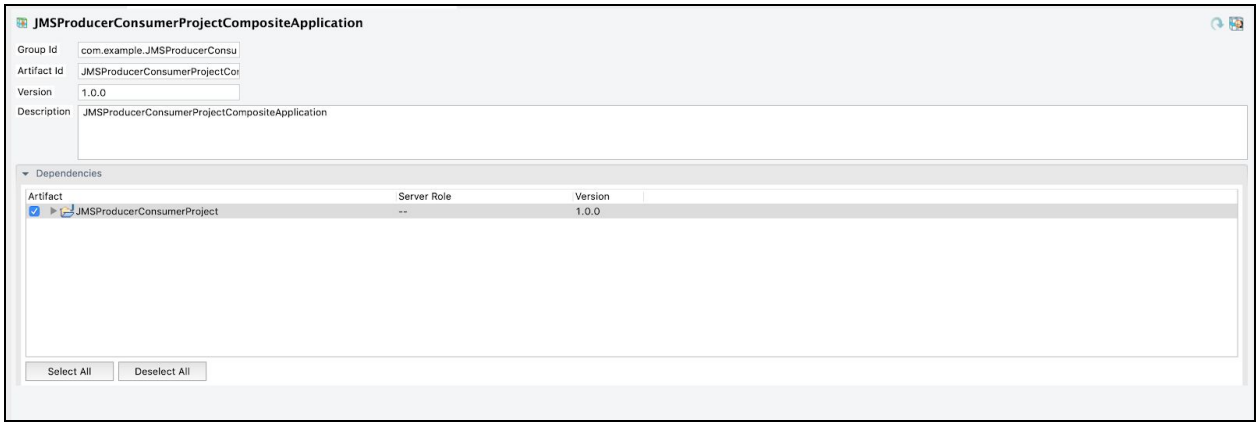
Property	Value
Url Style	URL_MAPPING
URL-Mappinh	/registerDoctor
Fault Sequence Type	Named Reference
Fault Sequence Name	faultSequence
In Sequence Type	Named Reference
In Sequence Name	RegisterDoctorInSequence
Methods	Tick on POST to enable



3. Save all your configurations.

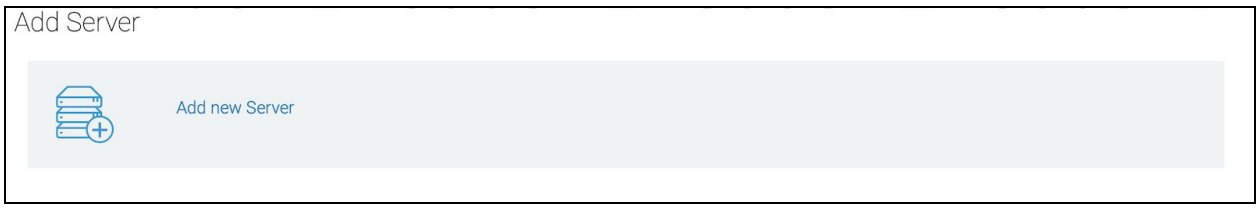
Create the Composite Application

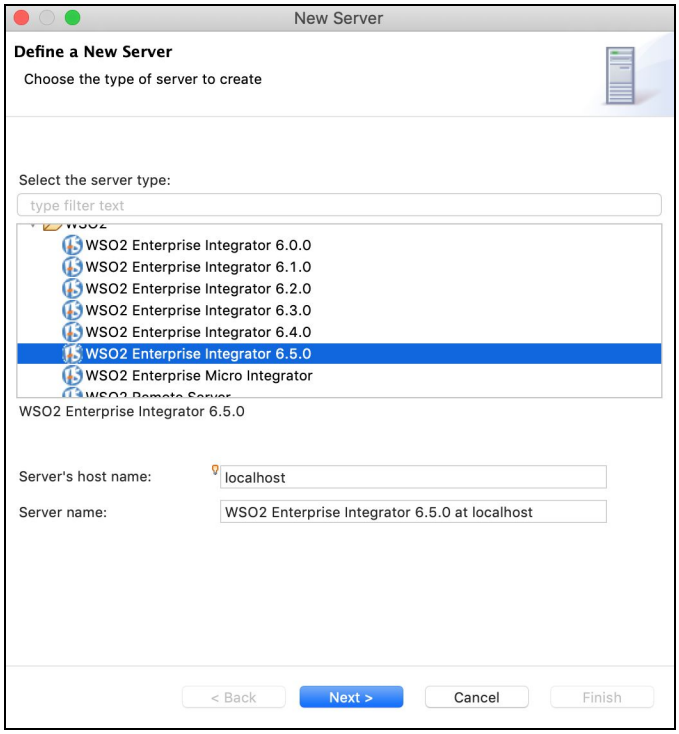
- Right click on the project and select **New** -> **Project -> Composite Application Project,** and create a Composite Application Project by selecting the `JMSProducerConsumerProject` project. Enter `JMSProducerConsumerCompositeApplication` for **Project Name**.



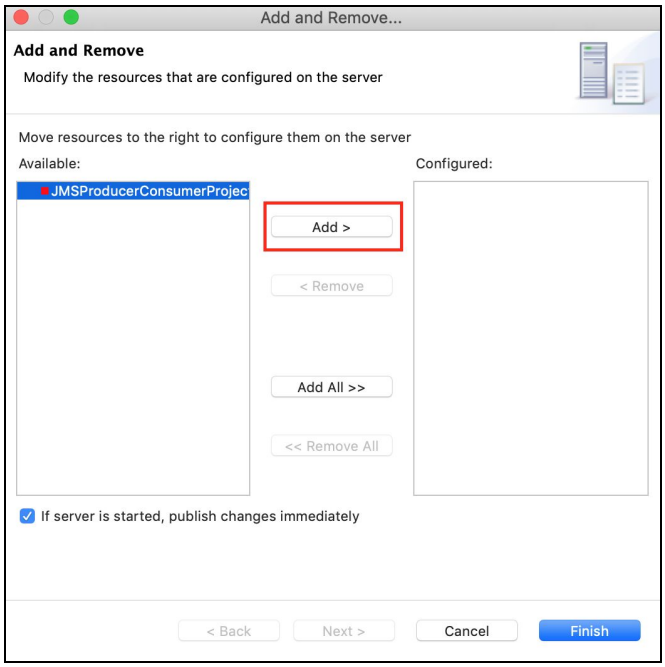
Test WSO2 EI as the JMS Producer and Consumer

1. Start the ActiveMQ Broker by executing `./activemq` console (if you did not already start it. If you are a Mac OS user, execute the following command: `sh activemq start`)
2. Click **Developer Studio ->Getting Started**, and then click the **Add New Server** link under **Add Server**.
3. Select **WSO2 Enterprise Integrator 6.5.0** and click **the Add** link in front of the **Server runtime environment** field.
4. Browse and locate the WSO2 Enterprise Integrator 6.1.1 distribution in your machine, and click **Next**.

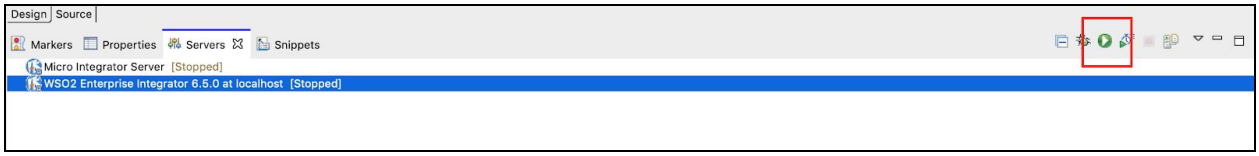




5. Select the Composite Application, click **Add** and click **Finish**.



6. Select the server you added, in the **Servers** tab and click the icon to run it.



7. Download the backend service (i.e. Hospital-Service-2.0.0.jar).
8. In a Terminal, navigate to the location where you saved the Hospital-Service-2.0.0.jar file and execute the following command to run it: `java -jar Hospital-Service-2.0.0.jar`
9. Create a new file named `add_doctor.json` with the following content:

```
{
      "name": "chris thomas",
      "hospital": "grand oak community hospital",
      "category": "gynaecology",
      "availability": "9.00 a.m - 11.00 a.m",
      "fee": "1900"
}
```

10. In a new tab of the Terminal, navigate to the location where you have the `add_doctor.json` file stored and execute the following command to send a request to the `HospitalServiceApi`: `curl -v -X POST`

```
"http://localhost:8280/hospitalservice/registerDoctor" --header
"Content-Type: application/json" -d @add_doctor.json -k -v
```

11. View the response on the Terminal. You will see a response similar to the following:

```
* Adding handle: conn: 0x7f8d61003a00
* Adding handle: send: 0
* Adding handle: recv: 0
* Curl_addHandleToPipeline: length: 1
* - Conn 0 (0x7f8d61003a00) send_pipe: 1, recv_pipe: 0
* About to connect() to localhost port 8280 (#0)
*   Trying ::1...
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 8280 (#0)
> POST /hospitalservice/registerDoctor HTTP/1.1
> User-Agent: curl/7.30.0
> Host: localhost:8280
> Accept: */*
> Content-Type: application/json
> Content-Length: 151
>
* upload completely sent off: 151 out of 151 bytes
< HTTP/1.1 202 Accepted
< Date: Thu, 04 May 2017 09:44:43 GMT
< Transfer-Encoding: chunked
<
* Connection #0 to host localhost left intact
```

12. View the response in the Developer Studio Console. You will see a response similar to the following:

```
[2017-05-04 15:13:53,650]  INFO - LogMediator To:
/hospitalservice/registerDoctor,MessageID:
urn:uuid:73f86800-0a4b-4c41-afc3-27a680b60df4,Direction:
request,register = ***********Registering Doctor ***********,Payload: {
      "name": "chris thomas",  "hospital": "grand oak community
hospital",  "category": "gynaecology",     "availability": "9.00 a.m -
11.00 a.m",  "fee": "1900"}
[2017-05-04 15:13:53,678]  INFO - TimeoutHandler This engine will expire
all callbacks after GLOBAL_TIMEOUT: 120 seconds, irrespective of the
timeout action, after the specified or optional timeout
[2017-05-04 15:13:56,129]  INFO - LogMediator To: ,MessageID:
ID:Praneeshas-MacBook-Air.local-64878-1493890774212-3:1:1:1:1,Direction:
request,query = *************Receiving Doctor Details *********,Payload:
{     "name": "chris thomas",  "hospital": "grand oak community
hospital",  "category": "gynaecology",     "availability": "9.00 a.m -
11.00 a.m",  "fee": "1900"}
[2017-05-04 15:13:56,767]  INFO - LogMediator To:
http://www.w3.org/2005/08/addressing/anonymous, WSAction: , SOAPAction:
, MessageID: urn:uuid:78710172-dc84-4592-8145-60a250b198c4, Direction:
request, Payload: {"status":"New Doctor Added Successfully"}
```

13. If you execute the cURL request again, you will see a response similar to the following.

```
[2017-05-04 15:14:43,863]  INFO - LogMediator To:
/hospitalservice/registerDoctor,MessageID:
urn:uuid:51968aad-9c01-439e-93a5-37ec9b16518f,Direction:
request,register = ***********Registering Doctor ***********,Payload: {
      "name": "chris thomas",  "hospital": "grand oak community
hospital",  "category": "gynaecology",     "availability": "9.00 a.m -
11.00 a.m",  "fee": "1900"}
[2017-05-04 15:14:45,094]  INFO - LogMediator To: ,MessageID:
ID:Praneeshas-MacBook-Air.local-64878-1493890774212-5:1:1:1:1,Direction:
request,query = *************Receiving Doctor Details *********,Payload:
{     "name": "chris thomas",  "hospital": "grand oak community
hospital",  "category": "gynaecology",     "availability": "9.00 a.m -
11.00 a.m",  "fee": "1900"}
[2017-05-04 15:14:45,116]  INFO - LogMediator To:
http://www.w3.org/2005/08/addressing/anonymous, WSAction: , SOAPAction:
, MessageID: urn:uuid:2b863fe8-94af-4248-aeb0-423497be0a21, Direction:
request, Payload: {"status":"Doctor Already Exist in the system"}
```

Lab: Custom Connector

Training Objective

Create a custom connector for a specific requirement that cannot be addressed via any of the existing connectors that can be downloaded from the connector store.

High Level Steps

- Download and install Apache Maven
- Create the Maven project template
- Add and configure files in the /src/main/resources directory
- Build the connector
- Upload the connector to the ESB profile of WSO2 Enterprise Integrator (WSO2 EI)
- Enable the connector and perform operations

Detailed Instructions

Create the Maven project template

1. Use the maven archetype to generate the Maven project template and sample connector code, run the following command in the directory where you want to create the connector on your local machine:

```
mvn org.apache.maven.plugins:maven-archetype-plugin:2.4:generate
-DarchetypeGroupId=org.wso2.carbon.extension.archetype
-DarchetypeArtifactId=org.wso2.carbon.extension.esb.connector-arch
etype -DarchetypeVersion=2.0.0
-DgroupId=org.wso2.carbon.esb.connector
-DartifactId=org.wso2.carbon.esb.connector.googlebooks
-Dversion=1.0.0
-DarchetypeRepository=http://maven.wso2.org/nexus/content/reposito
ries/wso2-public/
```

2. When prompted, enter a name for the connector.
   Note: Specify the name in upper camel case. For example, GoogleBooks.
3. When prompted for confirmation, enter `y`. The `org.wso2.carbon.esb.connector.googlebooks` directory is created with a directory structure consisting of a `pom.xml` file, `src` tree, and `repository` tree.

Add and configure files in the /src/main/resources directory

1. Create a directory named `googlebooks_volume` in `/src/main/resources`.
2. Create a file named `listVolume.xml` with the following content in the `googlebooks_volume` directory:

```
<template name="listVolume" xmlns="http://ws.apache.org/ns/synapse">
    <parameter name="searchQuery" description="Full-text search query
string." />
    <sequence>
      <property name="uri.var.searchQuery"
expression="$func:searchQuery" />
        <call>
            <endpoint>
                <http method="get"
uri-template="https://www.googleapis.com/books/v1/volumes?q={uri.var.sea
rchQuery}" />
            </endpoint>
        </call>
    </sequence>
</template>
```

3. Create a file named `component.xml` in the `googlebooks_volume` directory and add the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<component name="googlebooks_volume" type="synapse/template">
    <subComponents>
        <component name="listVolume">
            <file>listVolume.xml</file>
            <description>Lists volumes that satisfy the given
query.</description>
        </component>
    </subComponents>
</component>
```

4. Edit the connector.xml file in the src/main/resources directory and add a new dependency as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<connector>
    <component name="GoogleBooks" package="org.wso2.carbon.connector" >
```

```
      <dependency component="googlebooks_volume"/>
      <description>wso2 sample connector library</description>
      </component>
</connector>
```

5. Create a directory named `icon` in the `/src/main/resources` directory and add two icons.
    - You can check out icons from the following location: http://svn.wso2.org/repos/wso2/scratch/connectors/icons/

Build the connector

- Navigate to the `org.wso2.carbon.esb.connector.googlebooks` directory and run `mvn clean install`.

  This builds the connector and generates a zip file named `googlebooks.zip` in the `target` directory.

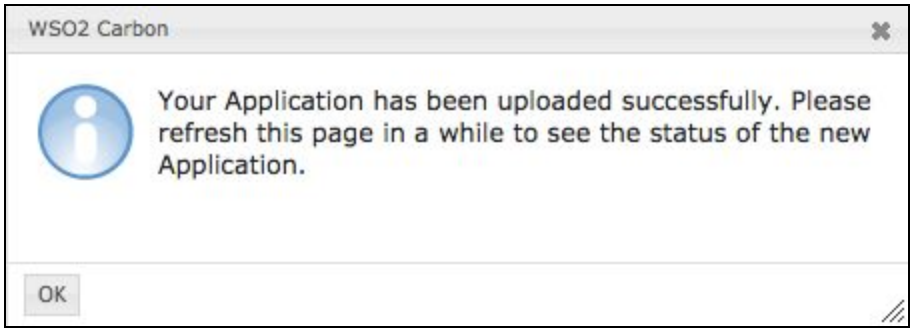
Upload the connector to the ESB Profile of WSO2 EI

Follow the steps below to upload the connector you created to the ESB profile of WSO2 EI.

1. Open a command prompt (or a shell in Linux), and go to the <EI_HOME>\bin directory.
2. Execute one of the following:
    - On Linux/Mac OS: `sh integrator.sh`
    - On Windows: `integrator.bat`

   The operation log keeps running until the ESB profile starts, which usually takes several seconds. Wait until the ESB profile fully boots up and displays a message similar to "WSO2 Carbon started in n seconds."

3. Open the Management Console of the ESB profile using https://localhost:9443/carbon, and log in using admin as the username as well as the password.
4. On the **Main** tab in the Management Console, under **Connectors** click **Add.** The **Add Connector** page opens.
5. On the **Add Connector** page, click **Choose File.**
6. Browse and select the `GoogleBooks-connector-1.0.0.zip` file, and then click **Upload**. On successful upload, you will see the following message.



7. Click **OK**. You will see that the connector is added to the list of all available connectors.

Enable the connector and perform various operations

1. Follow the steps below to enable the connector:
    a. On the **Main** tab in the Management Console, under **Connectors** click **List** to view the uploaded connector.
    b. Click **Enable** next to the connector that you uploaded.
2. Create the following proxy service:

```
<?xml version="1.0" encoding="UTF-8"?>
<proxy xmlns="http://ws.apache.org/ns/synapse"
       name="googlebooks_listVolume"
       transports="https,http"
       statistics="disable"
       trace="disable"
       startOnLoad="true">
   <target>
      <inSequence>
         <property name="searchQuery"
expression="json-eval($.searchQuery)"/>
         <googlebooks.listVolume>
            <searchQuery>{$ctx:searchQuery}</searchQuery>
         </googlebooks.listVolume>
         <respond/>
      </inSequence>
      <outSequence>
         <log/>
         <send/>
```

```
      </outSequence>
   </target>
   <description/>
</proxy>
```

3. Use a rest client and send the following request:

   Note: You can use Advanced REST client to send the request.

```
POST /services/googlebooks_listVolume HTTP/1.1
{
    "searchQuery":"aladin"
}
```

Alternatively, you can make a POST request with CURL from the command line as follows:

```
curl -v -X POST -d "{"searchQuery":"rabbit"}" -H
"Content-Type: application/json"
http://localhost:8280//services/googlebooks_listVolume
```

This performs a search and displays a list of volumes that meet the specified search criteria.

Lab: Custom Mediator

Training Objective

Create your own custom mediators to implement a specific business requirement that requires functionality not provided by the existing mediators. In WSO2 Enterprise Integrator (WSO2 EI)

High Level Steps

- Create the custom mediator using WSO2 EI Tooling
- Deploy the custom mediator in WSO2 EI
- Engage the custom mediator to the mediation flow

Detailed Instructions

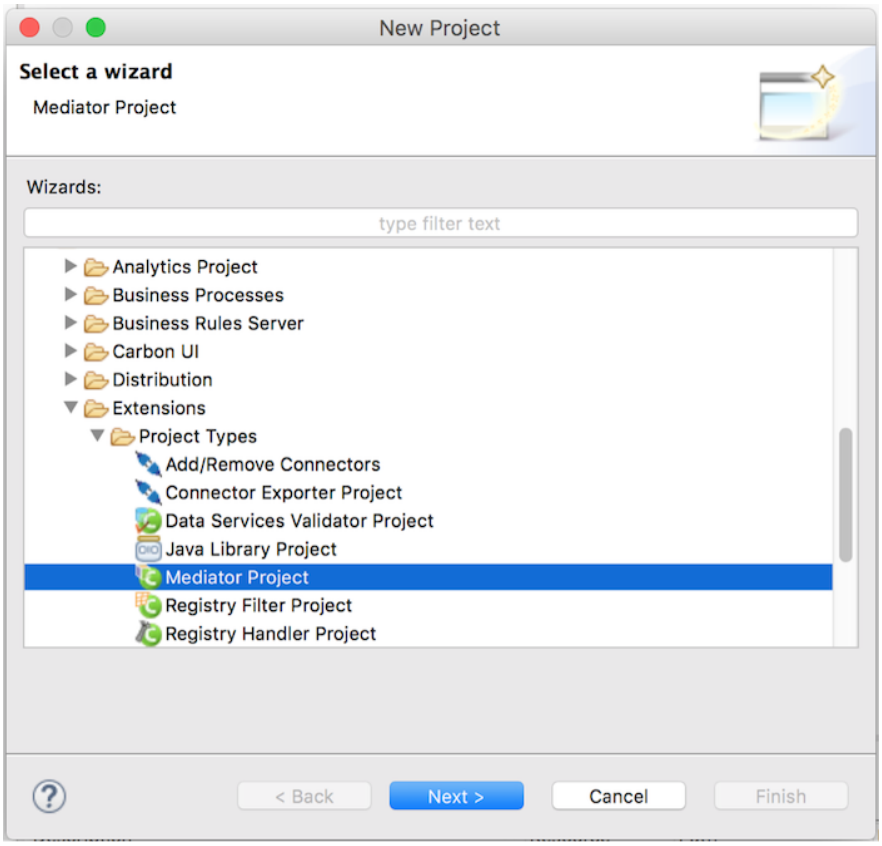
Create the custom mediator

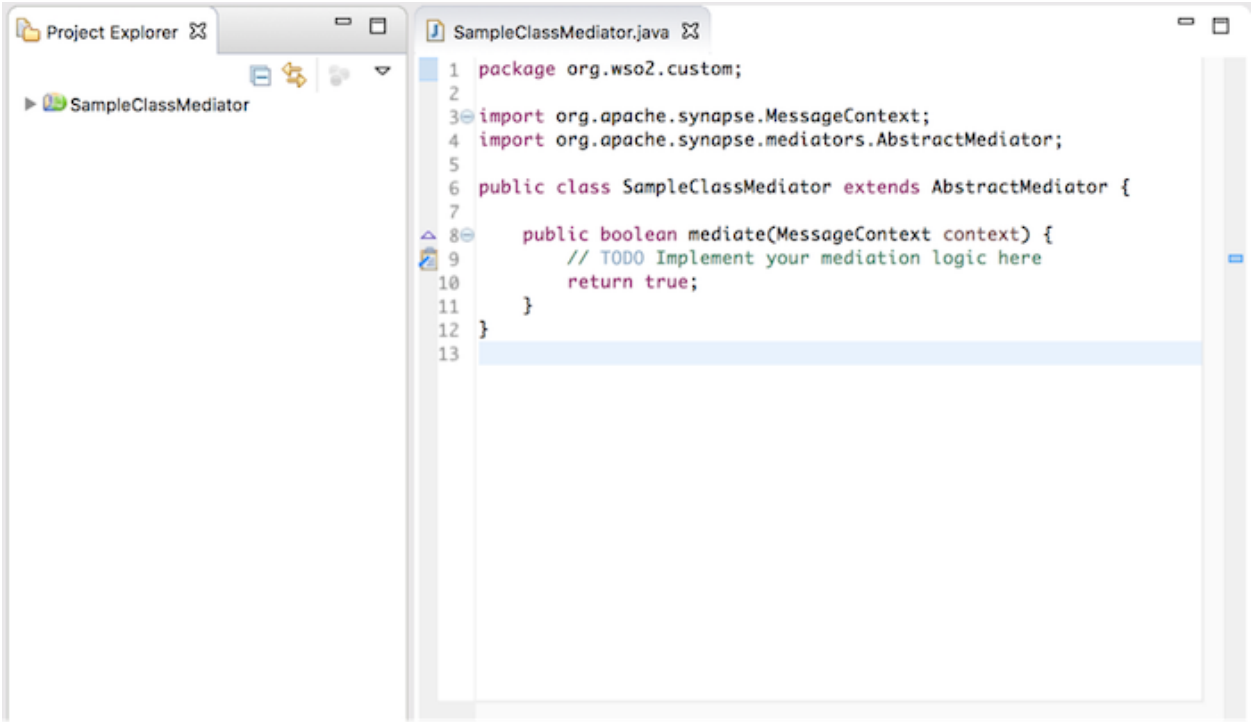
There are two ways you can follow when you want to create a custom mediator:

- Implement the mediator interface
- Extend `AbstractMediator` class

It is recommended to take the second approach because `AbstractMediator` provides all the common functionality you need.
To create a class mediator with the required project structure you can use WSO2 EI Tooling.

You can create a mediator project using WSO2 EI Tooling as illustrated in the following figure.





Following is the sample class mediator code:

```
package org.wso2.custom;

import org.apache.synapse.MessageContext;
import org.apache.synapse.mediators.AbstractMediator;

public class SampleClassMediator extends AbstractMediator {

      public boolean mediate(MessageContext synCtx) {
            // TODO Implement your mediation logic here
                        System.out.println("Hello, this is my first mediator
");
            return true;
      }
}
```

Here, we have extended the `AbstractMediator` class. The only method we need to implement is the mediate method, which is invoked when the mediator is executed at the mediation flow. The return statement of the mediate method decides whether the mediation flow should continue further or not.

Deploy the mediator

Follow one of the following procedures to deploy the custom mediator in WSO2 EI:

- Deploy as a server extension by copying the `.jar` file to the file system
- Pack and export as a composite application

Let's have a look at the detailed instructions on how to deploy the custom mediator you created as a server extension.

1. Export the custom mediator project that you created as a `.jar` file, and then copy it into the `<EI_HOME>/lib` directory.
   Note: If it is an OSGi bundle you can copy it into the `<EI_HOME>/dropins` directory.
2. Restart the ESB profile of WSO2 Integrator. This installs the new JAR.

When you copy the .jar file as a server extension, the custom mediator can be accessed from anywhere (even from the tenant space).

Now let's have a look at the detailed instructions on how to pack and export the custom mediator you created as a composite application

1. Use WSO2 EI tooling to pack the mediator project within a composite application and export it as a CAR file.
2. Deploy the CAR file in the ESB profile of WSO2 EI.
   When you deploy the CAR file in WSO2 EI, you can work with the mediator without having to restart the ESB profile of WSO2 EI.

Note: When you deploy the mediator through a CAR file, the mediator is accessible only to the artifacts (sequences, proxy services, APIs) available in the same CAR file. The mediator is not available globally. If you want to access the mediator from an artifact which is deployed from another CAR or from any other way, you can do so by following the steps below:

1. Write a sequence that engages the class mediator.
2. Pack the sequence from the same CAR file that contains the class mediator.
3. Call the sequence from other artifacts.

Engage the mediator to the message flow

To engage the custom class mediator into the message flow, you can use the built-in mediator called Class Mediator. Following is a sample configuration:

```
<class name="org.wso2.custom.SimpleClassMediator" />
```

Here, the name should be the full qualified class name of the mediator class.

Lab: Securing a Proxy Service

Training Objective

Create a security policy covering the security requirements, and apply the policy to a proxy service using WSO2 EI tooling.

High Level Steps

- Prerequisites
- Creating the security policy
- Add the security policy to the proxy service
- Deploying the secured proxy service in WSO2 EI

Detailed Instructions

https://docs.wso2.com/display/EI650/Applying+Security+to+a+Proxy+Service

Lab: Using Secure Vault

Training Objective

Use the **Secure Vault** implementation that is built in to the ESB profile to encrypt plain text passwords in configuration files and synapse configurations.

High Level Steps

- Encrypting passwords in configuration files
- Encrypting passwords for synapse configurations
- Using encrypted passwords in synapse configurations

Detailed Instructions

https://docs.wso2.com/display/EI650/Working+with+Passwords+in+the+ESB+profile

Lab: Streaming Files Using the VFS Transport

Training Objective

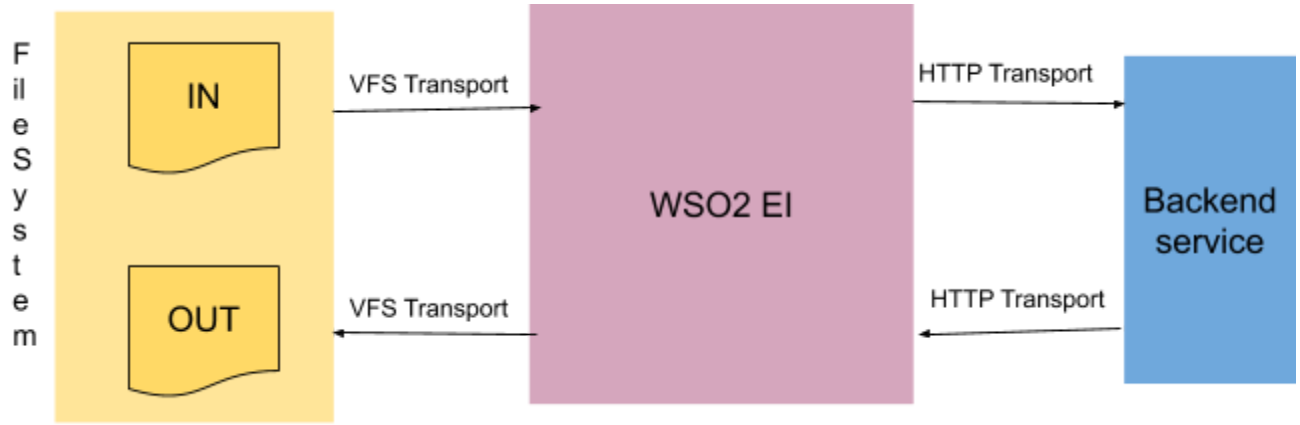
Use the Virtual File System (VFS) transport to stream files. The XML configuration for this sample (i.e., the *<EI_HOME>/samples/service-bus.synapse_sample_254.xml* file) is as follows:

```
<definitions xmlns="http://ws.apache.org/ns/synapse">
    <proxy name="StockQuoteProxy" transports="vfs">
        <parameter
name="transport.vfs.FileURI">file://C:\Users\user\in</parameter>
<!--CHANGE-->
        <parameter name="transport.vfs.ContentType">text/xml</parameter>
        <parameter name="transport.vfs.FileNamePattern">.*\.xml</parameter>
        <parameter name="transport.PollInterval">15</parameter>
        <parameter
name="transport.vfs.MoveAfterProcess">file://C:\Users\user\success</parameter>
<!--CHANGE-->
        <parameter
name="transport.vfs.MoveAfterFailure">file://C:\Users\user\failure</parameter>
<!--CHANGE-->
        <parameter name="transport.vfs.ActionAfterProcess">MOVE</parameter>
        <parameter name="transport.vfs.ActionAfterFailure">MOVE</parameter>
        <target>
            <endpoint>
                <address format="soap12"
uri="http://localhost:9000/services/SimpleStockQuoteService"/>
            </endpoint>
            <outSequence>
                <property name="transport.vfs.ReplyFileName"

expression="fn:concat(fn:substring-after(get-property('MessageID'),
'urn:uuid:'), '.xml')"
                           scope="transport"/>
                <property action="set" name="OUT_ONLY" value="true"/>
```

```
                <send>
                    <endpoint>
                        <address uri="vfs:file://C:\Users\user\out"/>
<!--CHANGE-->
                    </endpoint>
                </send>
            </outSequence>
        </target>
        <publishWSDL
uri="file:repository/samples/resources/proxy/sample_proxy_1.wsdl"/>
    </proxy>
</definitions>
```

The diagram below illustrates the flow of this use case.



High Level Steps

- Enabling the VFS transport
- Creating the VFS file locations
- Building the sample
- Executing the sample
- Analyzing the output

Detailed Instructions

Enabling the VFS transport

Uncomment the following VFS listener and the VFS sender configurations in the *<EI_HOME>/conf/axis2/axis2.xml* file.

```
<transportReceiver name="vfs"
class="org.apache.synapse.transport.vfs.VFSTransportListener"/>
...
<transportSender name="vfs"
class="org.apache.synapse.transport.vfs.VFSTransportSender"/>
```

Creating the VFS file locations

1. Create 4 new directories (folders) named '**in**', '**out', 'success'** and '**failure'** in a preferred location in a test directory (e.g., /home/user/test) in your local file system.
2. Open the *<EI_HOME>/samples/service-bus/synapse_sample_254.xml* file in a Text Editor.
3. Change the values of the following properties as follows:

Property	Value	Example
transport.vfs.FileURI	<location of the '**in**' directory you created>	*file://C:\Users\user\in*
transport.vfs.MoveAfterProcess	<location of the '**success**' directory you created>	*file://C:\Users\user\success*

transport.vfs.MoveAfterFailure	<location of the '**failure**' directory you created>	*file://C:\Users\user\failure*

4. Change the address URI of the endpoint in the out sequence to point to the out directory you created. For example,

```
<address uri="vfs:file://C:\Users\user\out"/>
```

Building the sample

Step 1: Start WSO2 EI

1. In a new Terminal, navigate to the *<EI_HOME>/bin* directory.
2. Execute the following command: *wso2ei-samples.bat -sn 254*

Step 2: Deploying the backend service

1. In another new Terminal, navigate to the *<EI_HOME>/samples/axis2Server/src/SimpleStockQuoteService* directory.
2. Execute the following command: *ant*

Step 3: Start the Axis2 server

1. In another new Terminal, navigate to the *<EI_HOME>/samples/axis2Server* directory.
2. Execute the following command: *axis2server.bat*

Executing the sample

1. Copy the *<EI_HOME>/samples/service-bus/resources/vfs/test.xml file.*
2. Paste in the location of the '**in**' directory you created before (e.g., *C:\Users\user\in*).

Analyzing the response

Verify the following output occurrences:

- The '**in**' directory should be empty.
- The '**success**' directory should have the *test.xml* file.
- The '**out**' directory should have the file received as the response from the backend service. For a sample response file, see below.

7af0339c-9867-4ddd-ad9f-17e1444f6045.xml

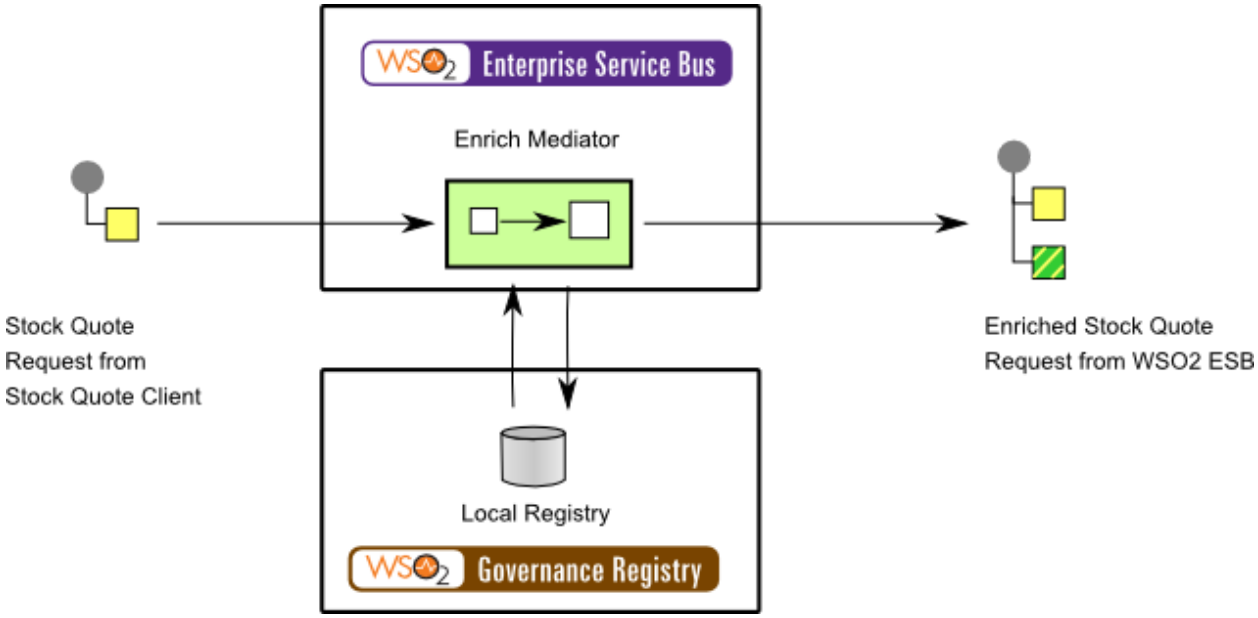

```xml
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
   <soapenv:Body>
      <ns:getQuoteResponse xmlns:ns="http://services.samples">
         <ns:return xmlns:ax21="http://services.samples/xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="ax21:GetQuoteResponse">
            <ax21:change>-2.6569888907121246</ax21:change>
            <ax21:earnings>-8.369238828340784</ax21:earnings>
            <ax21:high>98.2010906845192</ax21:high>
            <ax21:last>94.86093794044126</ax21:last>
            <ax21:lastTradeTimestamp>Fri May 18 04:25:47 IST
2018</ax21:lastTradeTimestamp>
            <ax21:low>-93.24479419506925</ax21:low>
            <ax21:marketCap>5.44331769277066E7</ax21:marketCap>
            <ax21:name>IBM Company</ax21:name>
            <ax21:open>-93.12814959185145</ax21:open>
            <ax21:peRatio>24.353901856986624</ax21:peRatio>
            <ax21:percentageChange>2.970719400743781</ax21:percentageChange>
            <ax21:prevClose>-89.4392412170228</ax21:prevClose>
            <ax21:symbol>IBM</ax21:symbol>
            <ax21:volume>6208</ax21:volume>
         </ns:return>
      </ns:getQuoteResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

Lab: Enriching Message Content

Training Objective

The Content Enricher EIP facilitates communication with another system if the message originator does not have all the required data items available. It accesses an external data source to augment a message with missing information. For more information, refer to the EIP Documentation.

This example scenario depicts a stock quote service. The client sends a stock quote request to WSO2 EI with only an identity number. However, in order to provide a stock quote, the sample Axis2 server at the back-end needs to map the identity number with a corresponding name, which is in an external source. The values are stored in the registry as a local entry. When the request arrives, the identity will be analyzed using the Switch mediator. Sequentially, the identity number will be replaced with the local entry using the Enrich mediator.



High Level Steps

- Starting the Axis2 server
- Deploying the backend service
- Starting WSO2 EI
- Creating the artifacts
- Sending the request
- Analyzing the response

Detailed Instructions

Deploying the backend service

1. In another new Terminal, navigate to the *<EI_HOME>/samples/axis2Server/src/SimpleStockQuoteService* directory.
2. Execute the following command: *ant*

Starting the Axis2 server

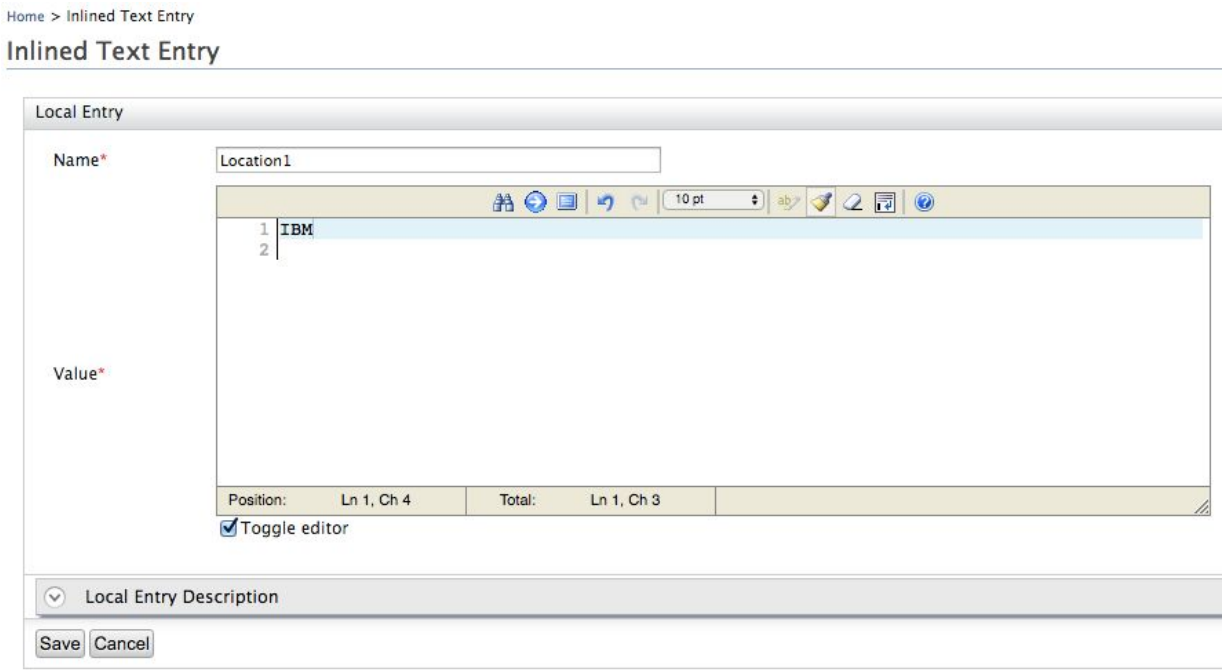
1. In another new Terminal, navigate to the *<EI_HOME>/samples/axis2Server* directory.
2. Execute the following command: *axis2server.bat*

Starting WSO2 EI

1. In another new Terminal, navigate to the *<EI_HOME>/bin* directory.
2. Execute the following command: *integrator.bat*

Creating the artifacts

1. Log in to the Management Console of the ESB profile using admin/admin credentials.
2. In the Management Console, click **Local Entries** and click **Add In-lined Text Entry**.
3. Enter *Location1* for **Name** and *IBM* for **Value**.



4. Click **Save**.
5. Once again, in the Management Console, click **Local Entries** and click **Add In-lined Text Entry**.
6. Click **Add Local Entries** tab and then click **Add In-lined Text Entry**.
7. Enter *Location2* for **Name** and *WSO2* for **Value**.
8. Click **Save**.
9. Click **Proxy service**, and then click **Custom Proxy**.
10. Click **switch to source view**.
11. Replace the content with the following:

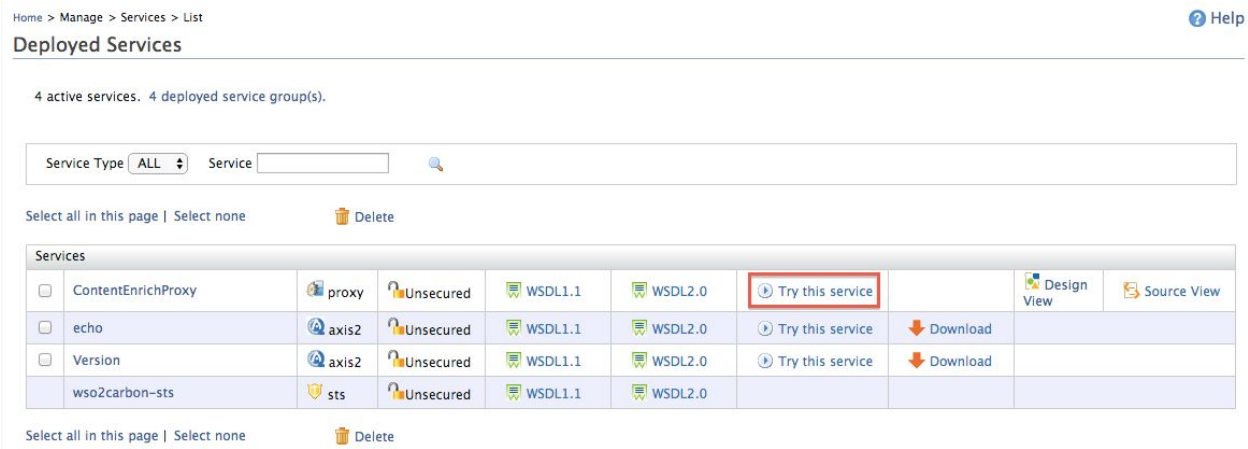
```
<?xml version="1.0" encoding="UTF-8"?>
<proxy xmlns="http://ws.apache.org/ns/synapse"
      name="ContentEnrichProxy"
      startOnLoad="true"
      statistics="disable"
      trace="disable"
      transports="http,https">
  <target>
    <inSequence>
      <switch xmlns:m0="http://services.samples"
            xmlns:m1="http://services.samples/xsd"
            source="//m1:symbol">
        <case regex="1">
          <log level="full"/>
          <enrich>
            <source clone="true" key="Location1" type="inline"/>
```

```
            <target xpath="//m1:symbol/text()"/>
          </enrich>
        </case>
        <case regex="2">
          <enrich>
            <source clone="true" key="Location2" type="inline"/>
            <target xpath="//m1:symbol/text()"/>
          </enrich>
        </case>
      </switch>
      <send>
        <endpoint>
          <address uri="http://localhost:9000/services/SimpleStockQuoteService"/>
        </endpoint>
      </send>
    </inSequence>
    <outSequence>
      <send/>
    </outSequence>
  </target>
  <publishWSDL preservePolicy="false"
            uri="file:samples/service-bus/resources/proxy/sample_proxy_1.wsdl"/>
  <description/>
</proxy>
```

12. Click **Save**.

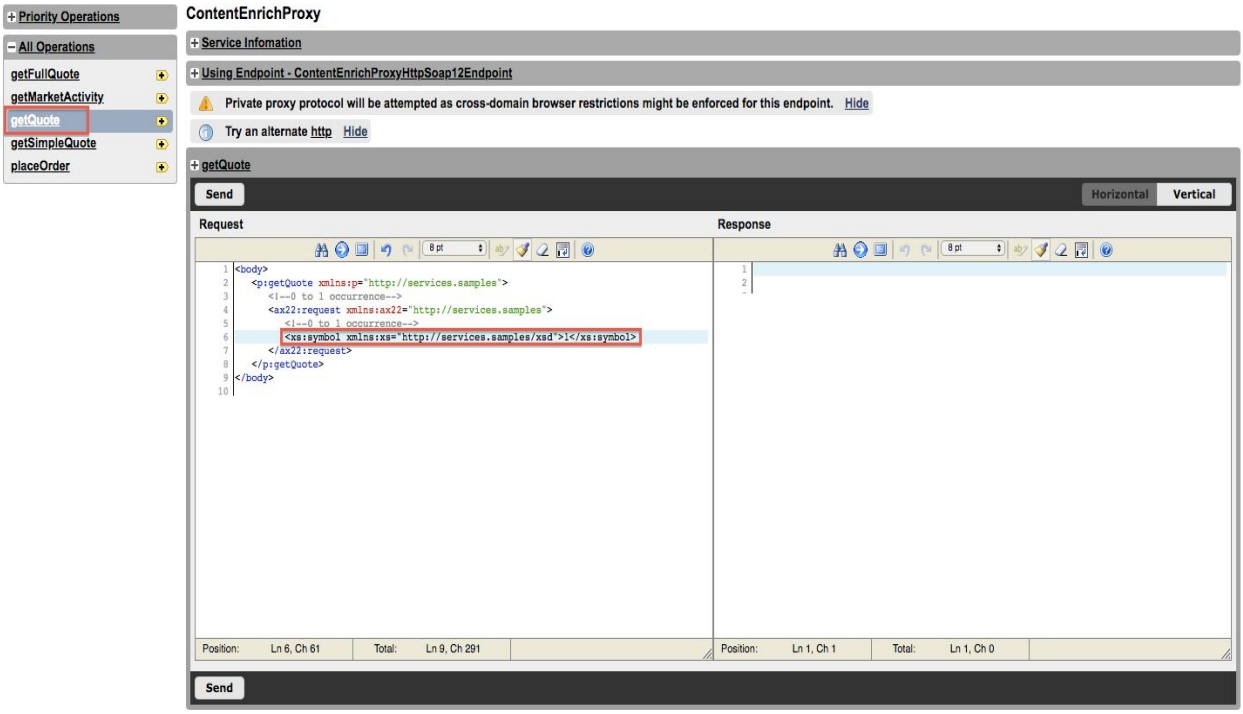
Sending the request

1. In the Management Console, click **List** under Services.
2. Click the **Try this service** link of the **ContentEnrichProxy**.



3. In the TryIt Tool, click **GetQuote**.
4. Enter 1 as the value of the request by replacing the question mark as follows:

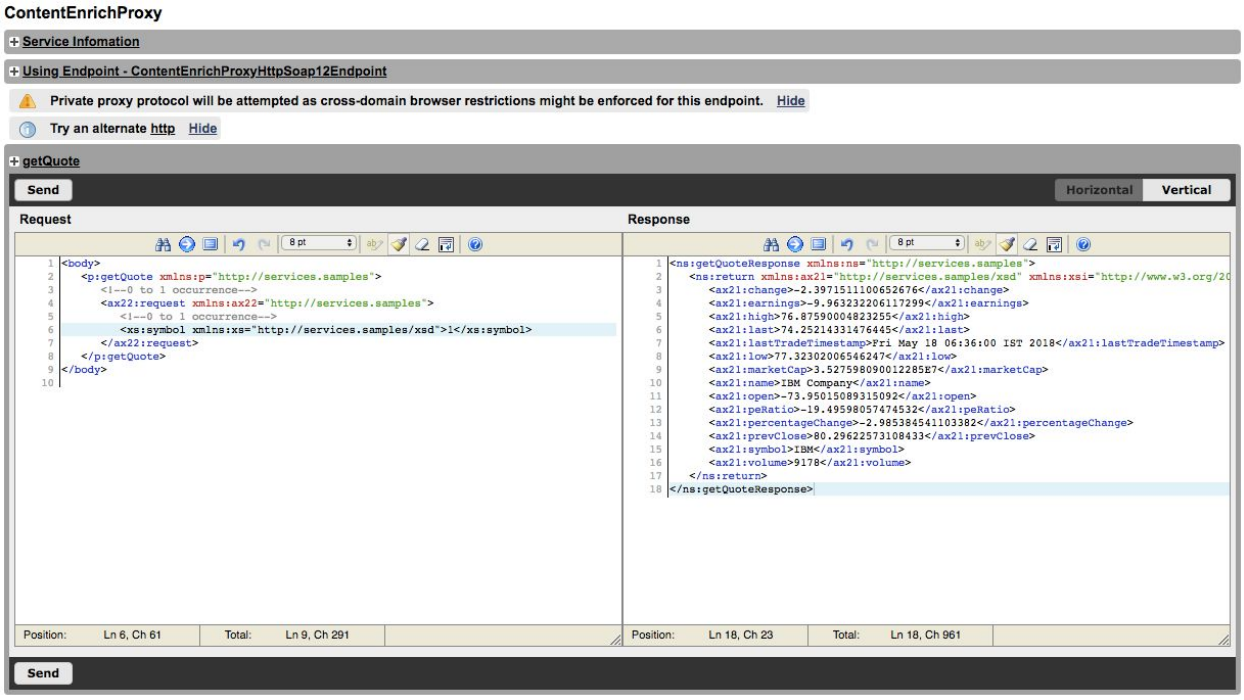
```
<xs:symbol xmlns:xs="http://services.samples/xsd">1</xs:symbol>
```



5. Click **Send**.

Analyzing the response

You view the response as shown below.



.

```
<ns:getQuoteResponse xmlns:ns="http://services.samples">
  <ns:return xmlns:ax21="http://services.samples/xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="ax21:GetQuoteResponse">
    <ax21:change>-2.3971511100652676</ax21:change>
    <ax21:earnings>-9.963232206117299</ax21:earnings>
    <ax21:high>76.87590004823255</ax21:high>
    <ax21:last>74.25214331476445</ax21:last>
    <ax21:lastTradeTimestamp>Fri May 18 06:36:00 IST 2018</ax21:lastTradeTimestamp>
    <ax21:low>77.32302006546247</ax21:low>
    <ax21:marketCap>3.527598090012285E7</ax21:marketCap>
    <ax21:name>IBM Company</ax21:name>
    <ax21:open>-73.95015089315092</ax21:open>
    <ax21:peRatio>-19.49598057474532</ax21:peRatio>
    <ax21:percentageChange>-2.985384541103382</ax21:percentageChange>
    <ax21:prevClose>80.29622573108433</ax21:prevClose>
    <ax21:symbol>IBM</ax21:symbol>
    <ax21:volume>9178</ax21:volume>
  </ns:return>
</ns:getQuoteResponse>
```