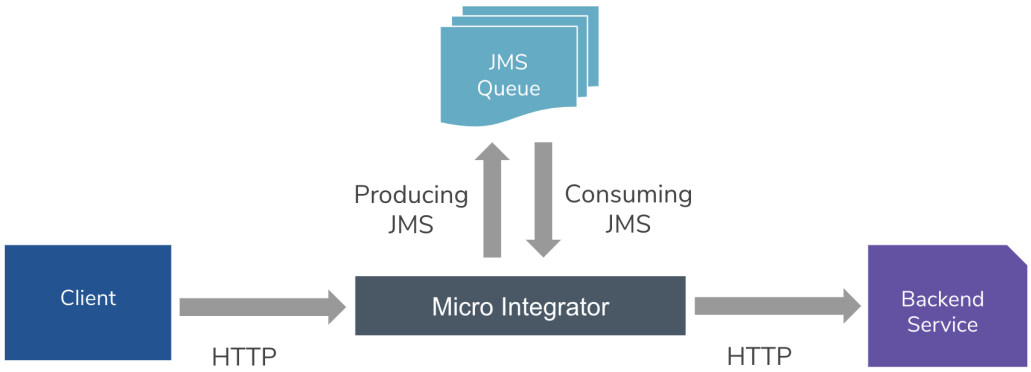
Lab: Micro Integrator as a JMS Producer and Consumer

Training Objective

Let's configure ActiveMQ with the Micro Integrator to demonstrate how it can function as a JMS producer as well as a consumer.



With the JMS transport, you can decouple the message sender and receiver. This means that the message sender can send messages reliably regardless of whether the system that receives the messages is running during that particular time.

In this scenario, an HTTP client sends a request to register a new doctor to the sample Healthcare service through the Micro Integrator. The Micro Integrator (working as the JMS producer) then queues the message to the JMS queue in the ActiveMQ broker. Once messages are queued, the Micro Integrator (now working as the JMS consumer) can connect to the queue and asynchronously consume the message/s.

High Level Steps

- Configure Apache ActiveMQ.
- Create the integration artifacts using WSO2 Integration Studio.
    - Endpoint artifact
    - REST API (JMS producer)
    - Inbound endpoint (JMS sender)
    - Mediation sequences
- Test the scenario.

Detailed Instructions

Creating the Integration Artifacts

Let's develop the integration artifacts using WSO2 Integration Studio.

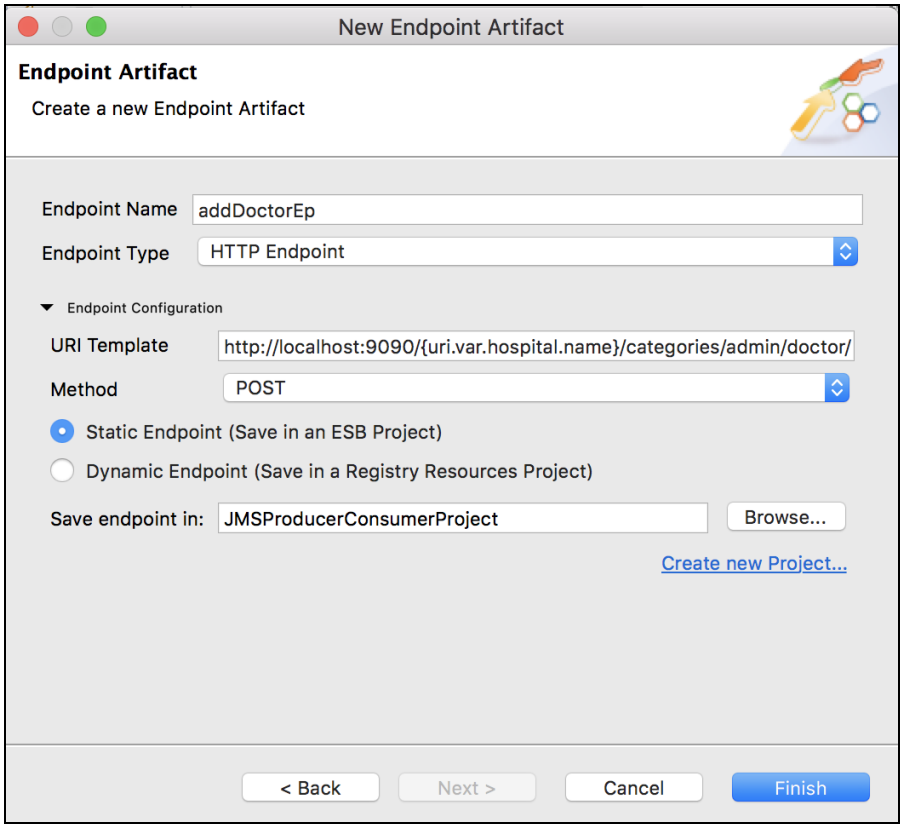
Creating the Project

1. Install WSO2 Integration Studio.
2. Open **WSO2 Integration Studio** and create an **Integration Project** named `JMSProducerConsumerProject`.

Creating the Endpoint

Right-click the ESB Config module in the navigator and select **New** -> **Endpoint** to create a new HTTP Endpoint named `addDoctorEp`. This will be used to send the doctor details to the back-end service (Healthcare service).

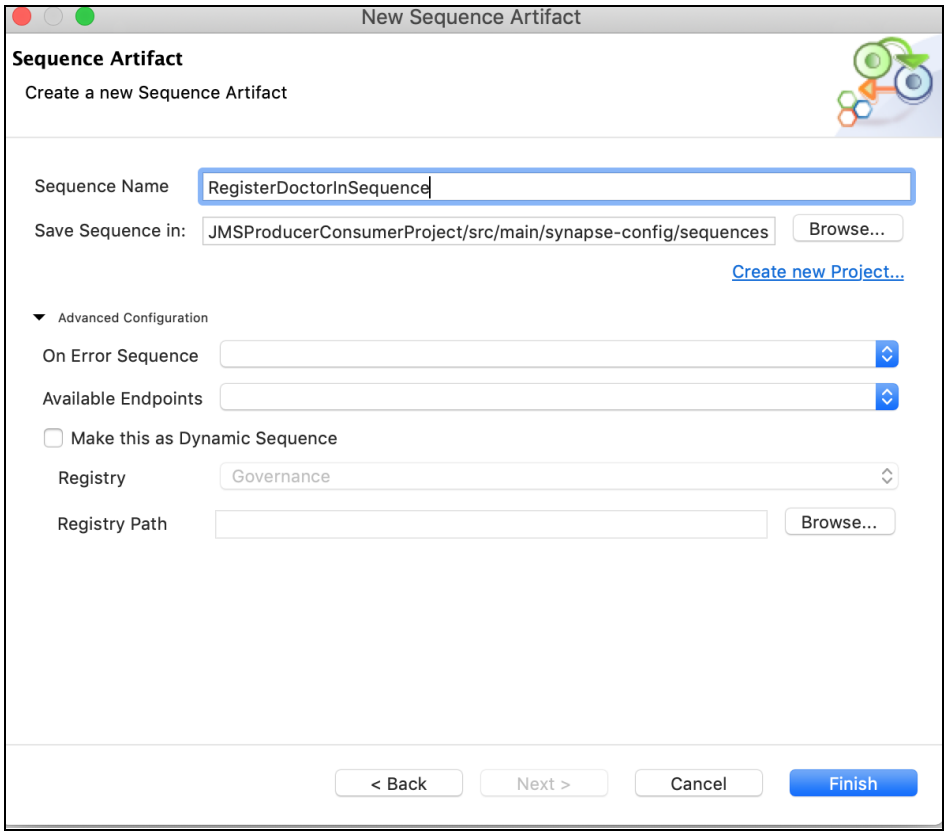
Field	Value
Endpoint Name	addDoctorEp
Endpoint Type	HTTP Endpoint
URI Template	http://localhost:9090/{uri.var.hospital.name}/categories/admin/doctor/newdoctor
Method	POST



Creating sequence: RegisterDoctorInSequence

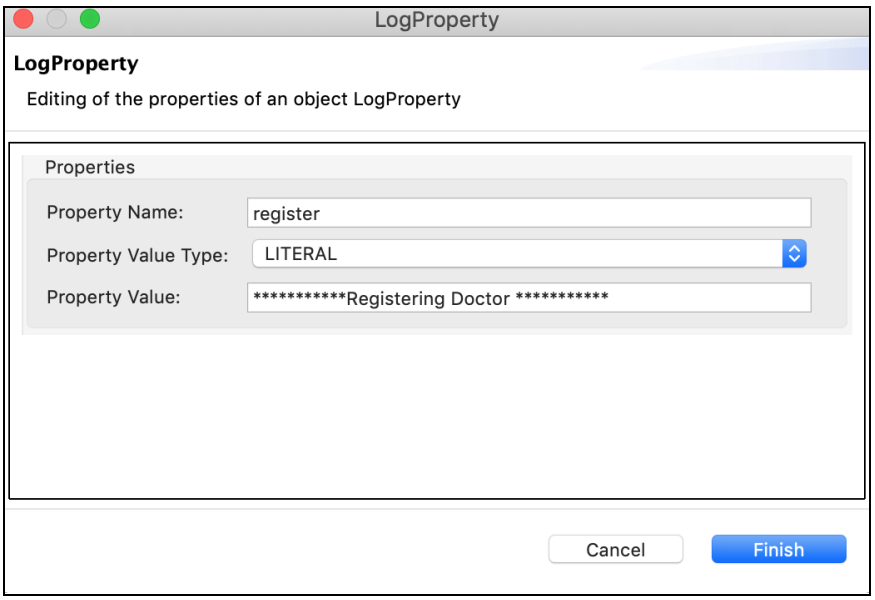
This sequence will be used to send the doctor details to the JMS queue when the request is received.

1. Right-click the ESB Config module and select **New** -> **Sequence** to create a new sequence named `RegisterDoctorInSequence`.



2. In the **Design View** of WSO2 Integration Studio, drag and drop a **Log** mediator to the `RegisterDoctorInSequence` sequence and change its properties as follows:

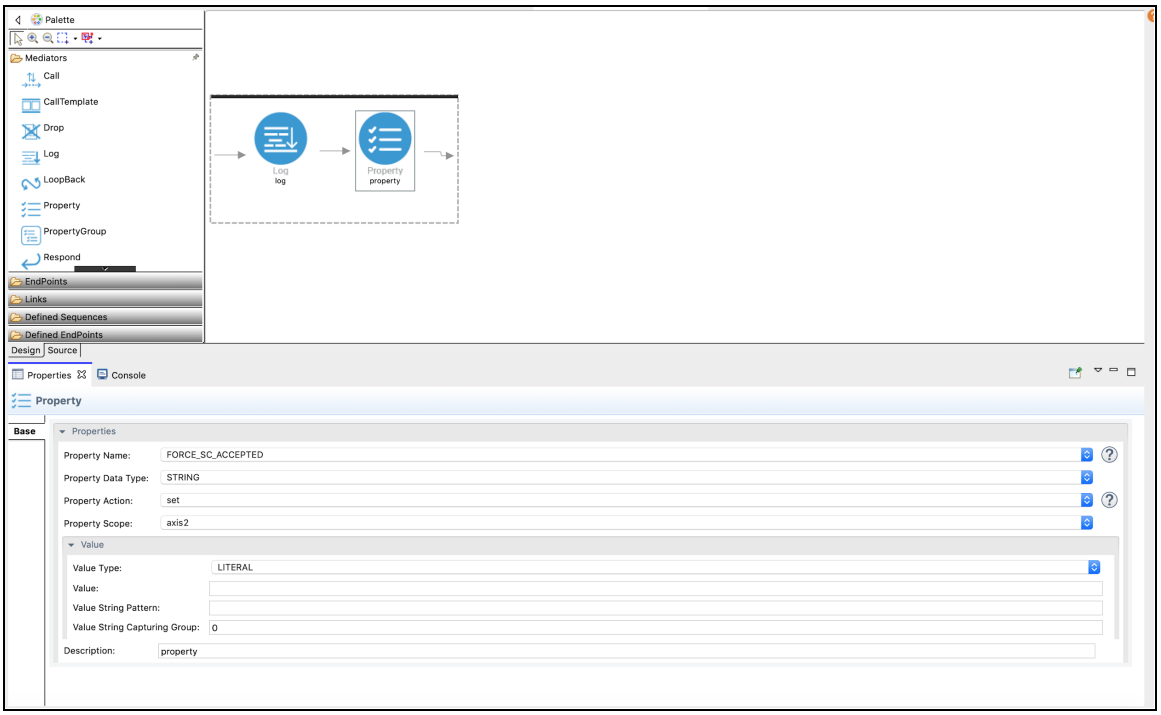
Property	Value
Log Category	INFO
Log Level	FULL
Log Separator	,
Properties	Select the '+' icon to open the LogProperty dialog box:



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

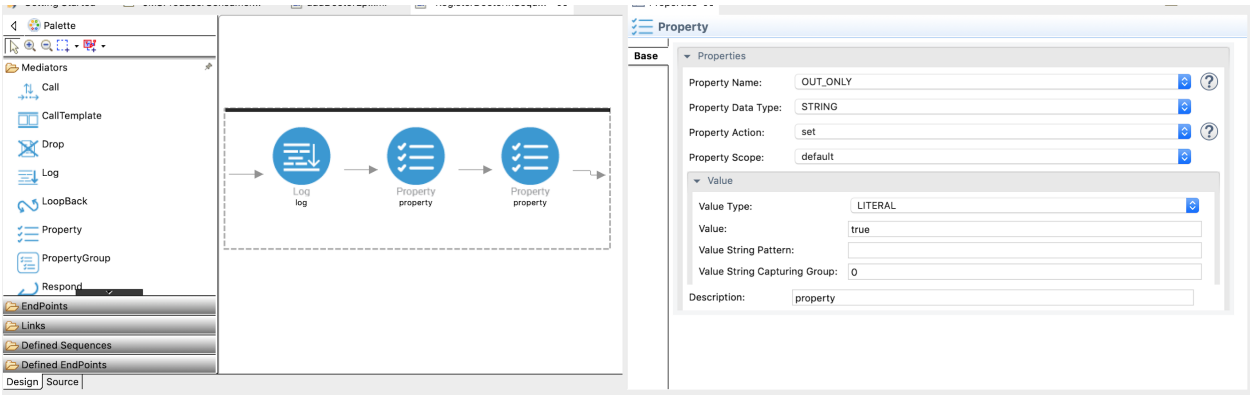
3. Drag and drop a **Property** mediator to the `RegisterDoctorInSequence` sequence and change its properties as follows:

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

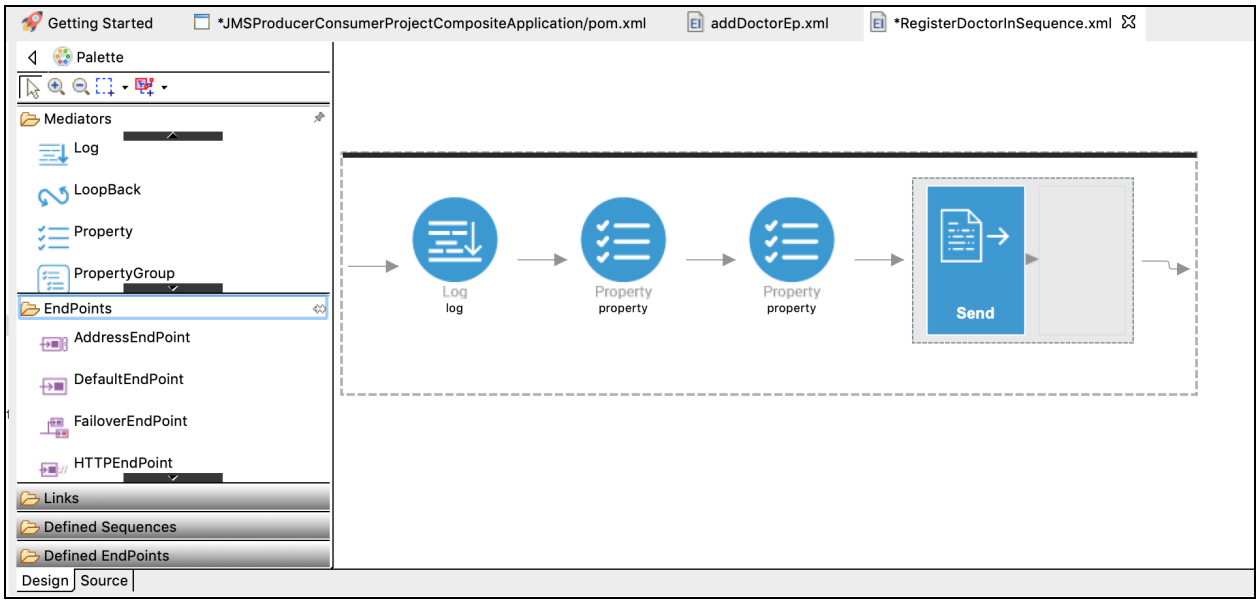


4. Insert another **Property** mediator to the `RegisterDoctorInSequence` sequence and change its properties as follows:

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:

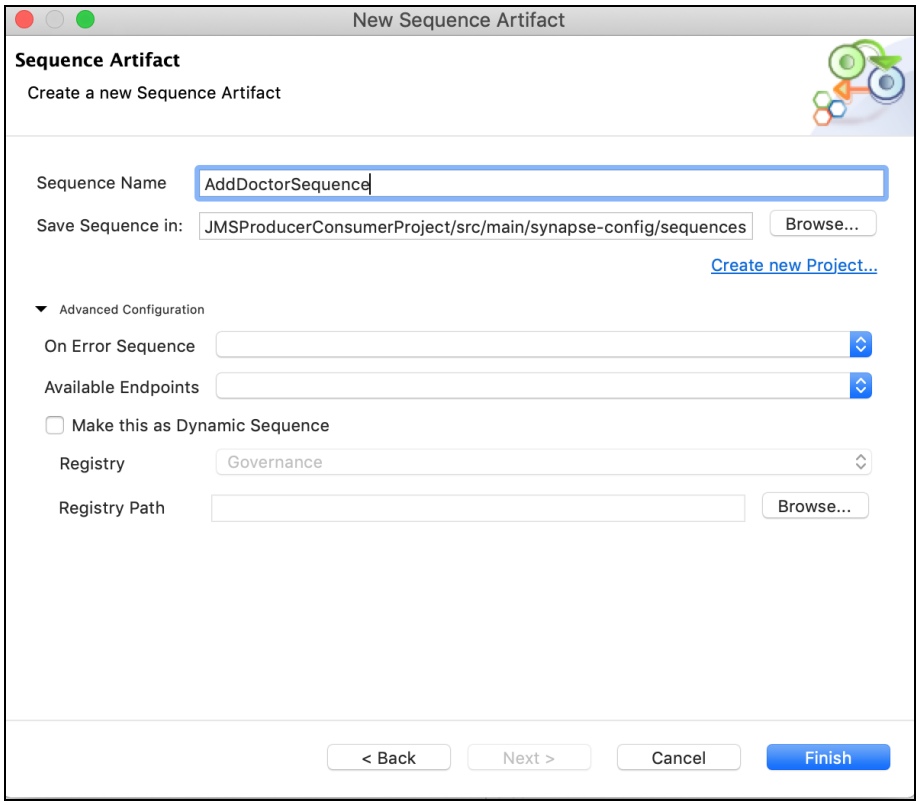
Property	Value
URI	jms:/RegisterDoctorService?transport.jms.ConnectionFactoryJNDIName=QueueConnectionFactory&java.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory&java.naming.provider.url=tcp://localhost:61616&transport.jms.DestinationType=queue

***NOTE**: WSO2 Integration Studio will escape & > &*

Creating sequence: `AddDoctorSequence`

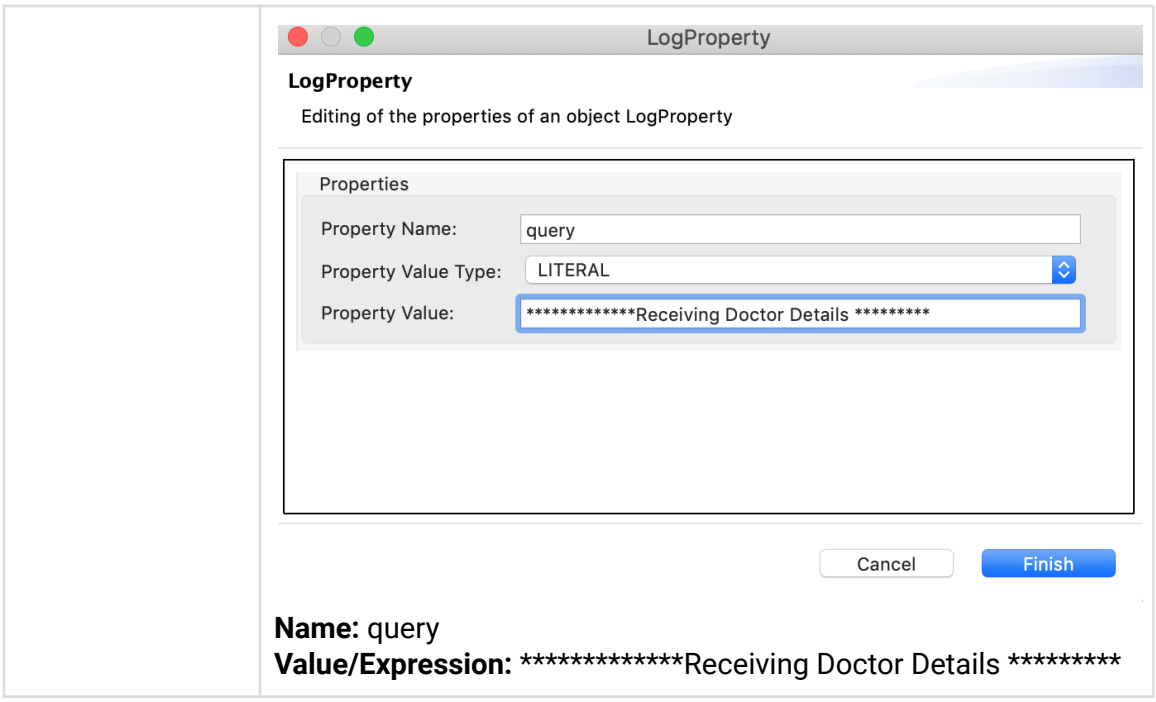
This will process the doctor details received from the JMS queue before sending to the back-end service (Healthcare service).

1. Right-click the project and select **New** -> **Sequence** to create a new sequence named `AddDoctorSequence`.

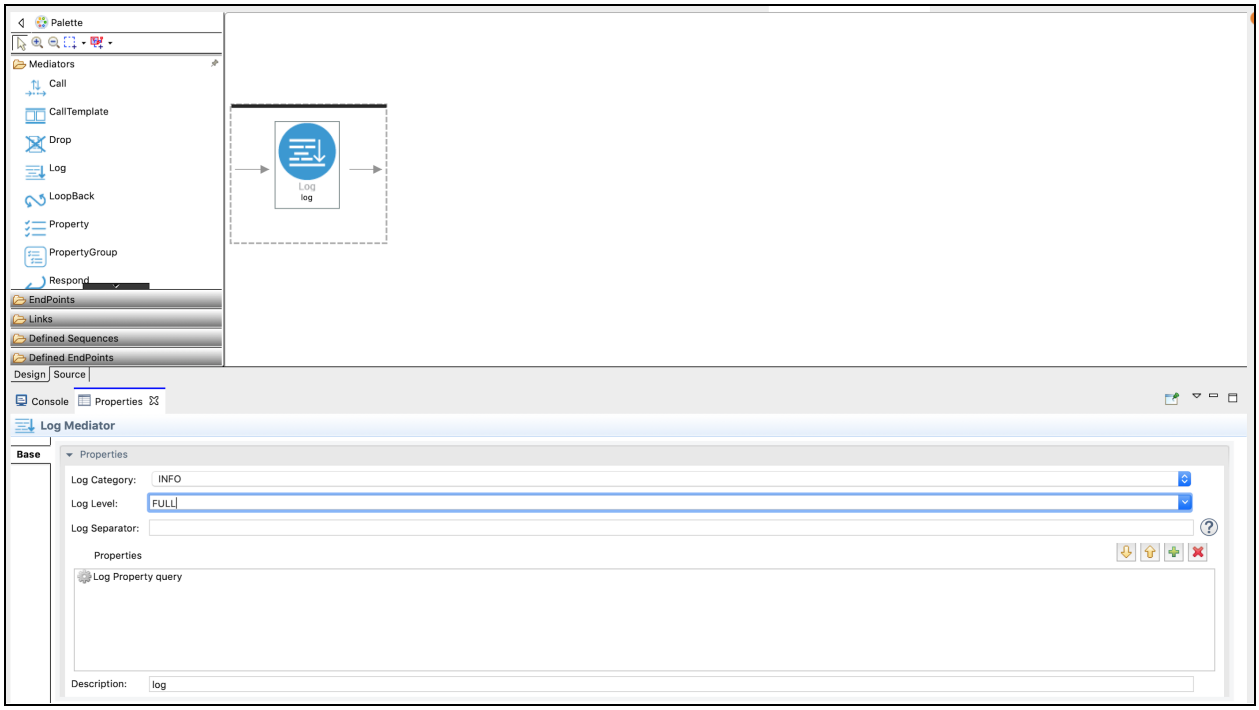


2. In the **Design View** of WSO2 Integration Studio, drag and drop a **Log** mediator to the `AddDoctorSequence` sequence and change its properties as follows:

Property	Value
Log Category	INFO
Log Level	FULL
Log Separator	,
Properties	Click the '+' icon to open the LogProperty dialog box:

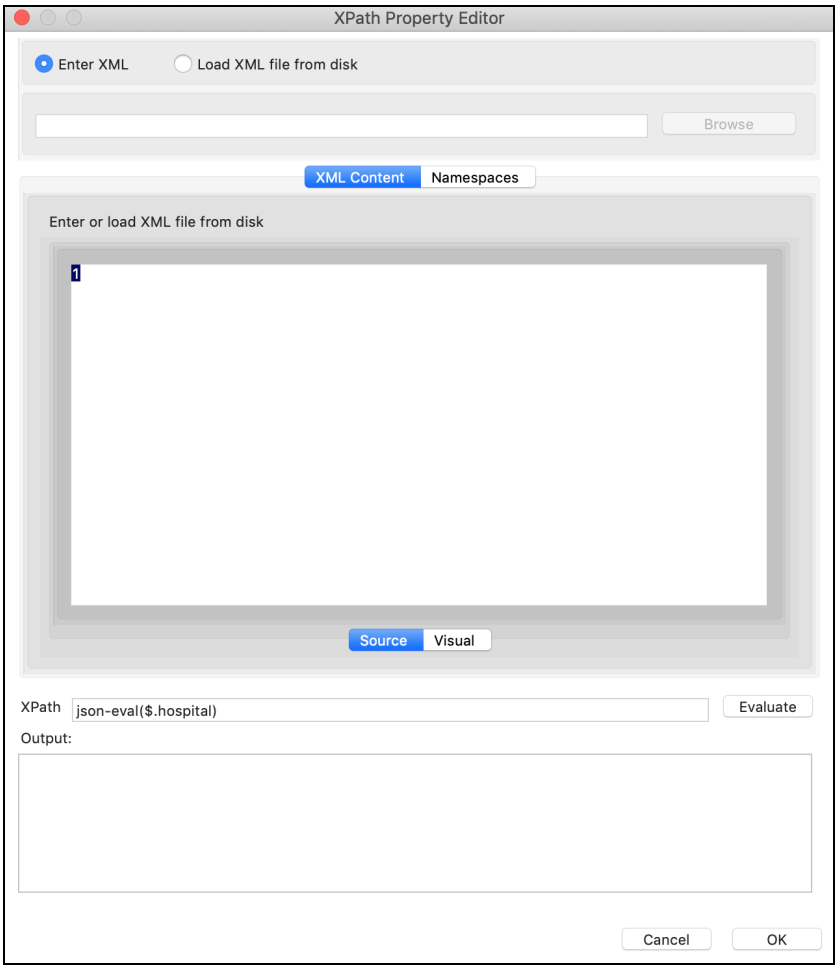


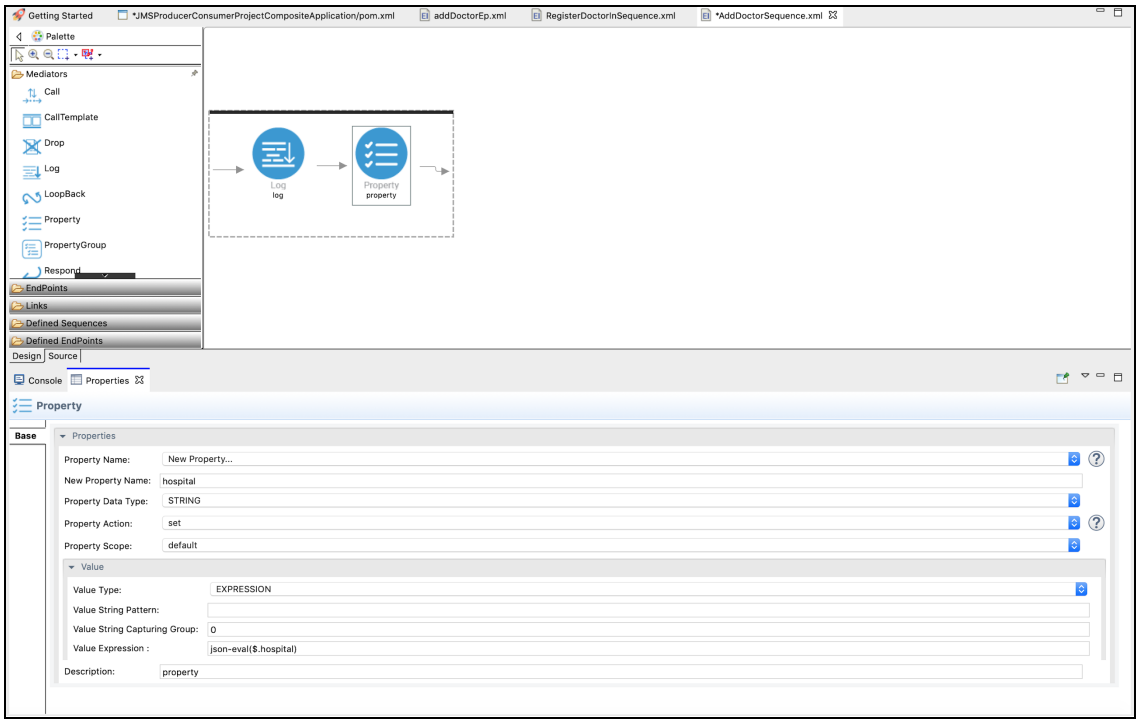
Name: query
Value/Expression: **********Receiving Doctor Details *********



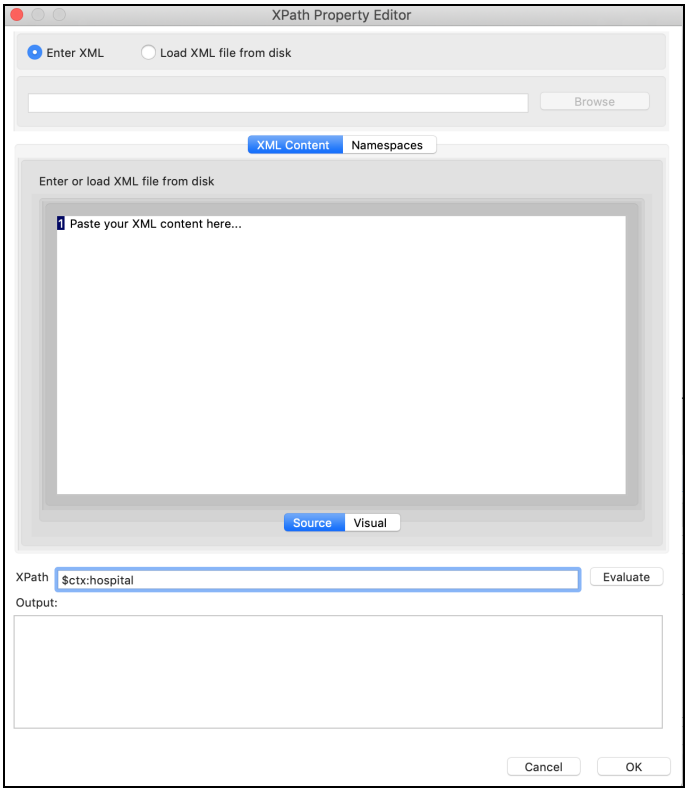
3. Drag and drop a **Property** mediator to the `AddDoctorSequence` sequence and change its properties as follows:

Property	Value

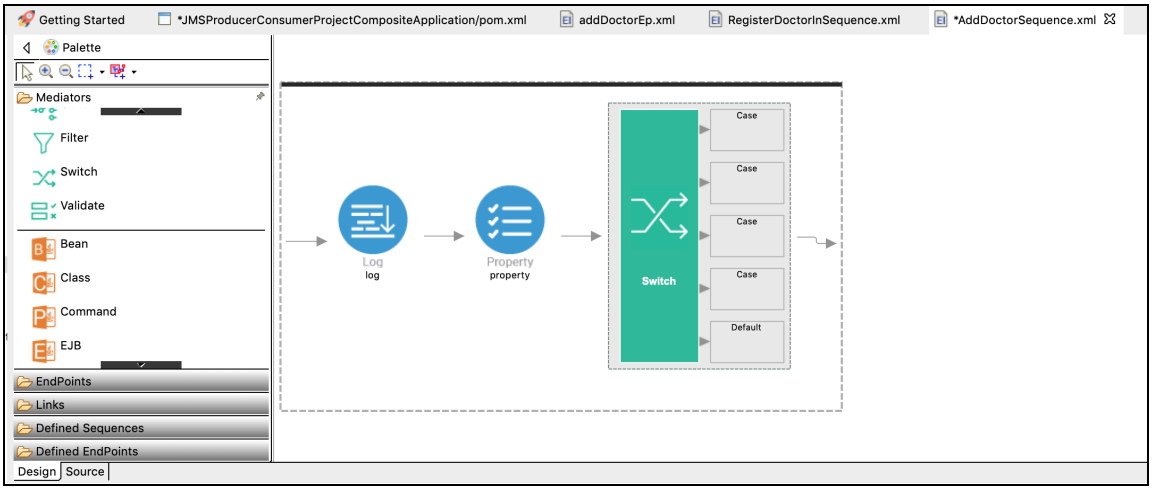
Property Name	New Property
New Property Name	hospital
Value Type	EXPRESSION
Value Expression	Click on the value field and enter the json-eval($.hospital) expression as follows: 



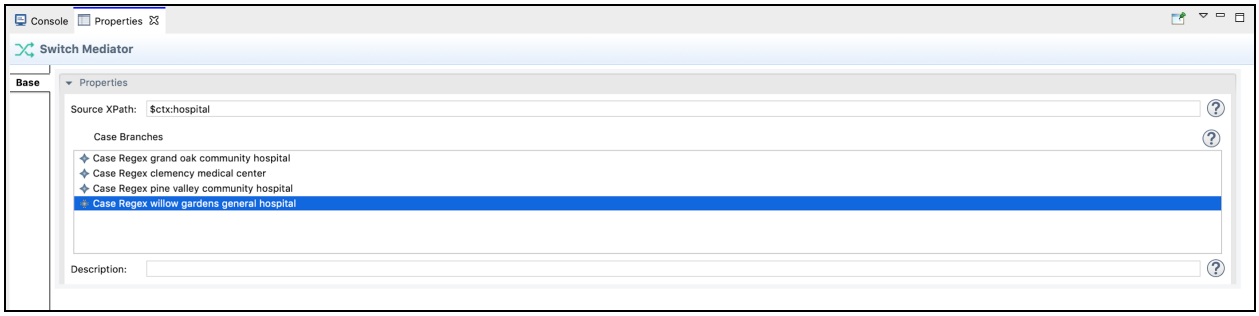
4. Drag and drop a **Switch** mediator to the `AddDoctorSequence` sequence.
5. In the **Properties** tab of the **Switch** mediator, click the value field of the **Source Xpath** and enter the following Xpath value: `$ctx:hospital`



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



7. Double-click 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



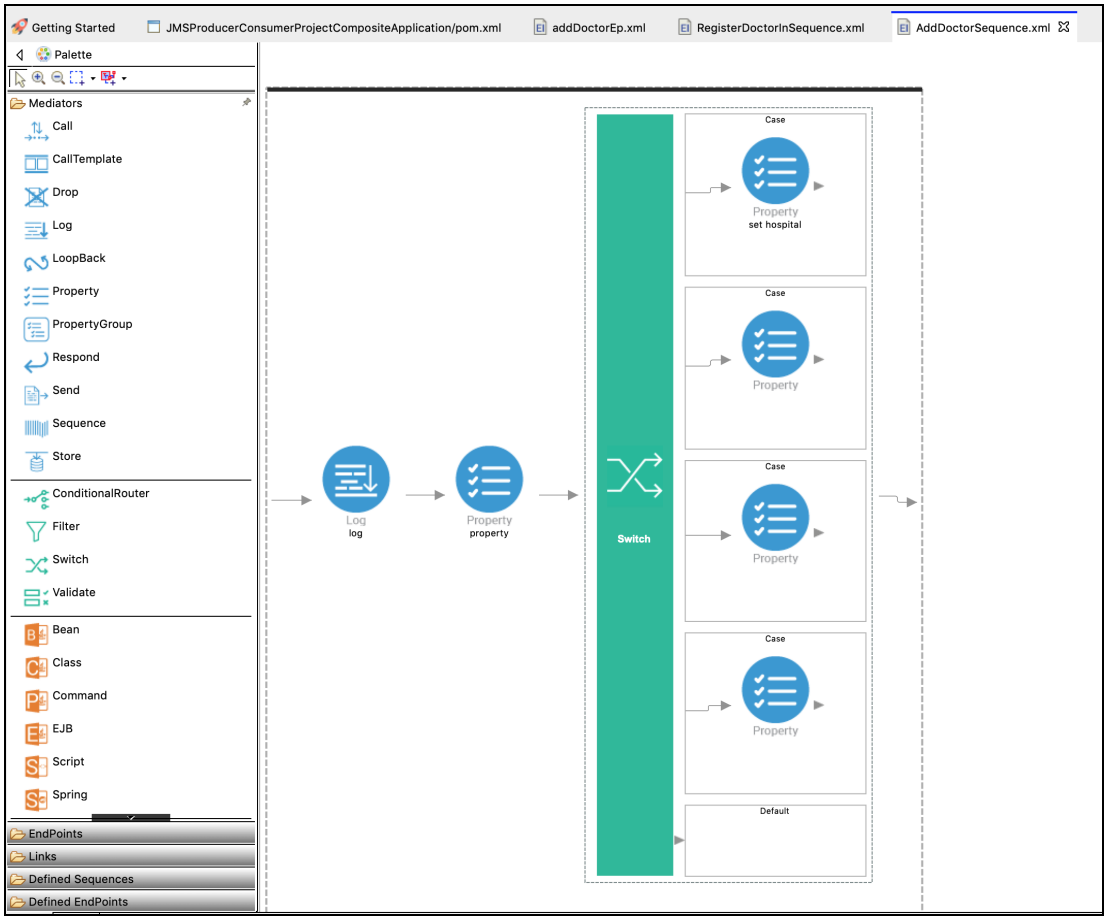
8. Drag and drop a **Property** mediator to the first case and change its properties as follows:

Case 1:

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

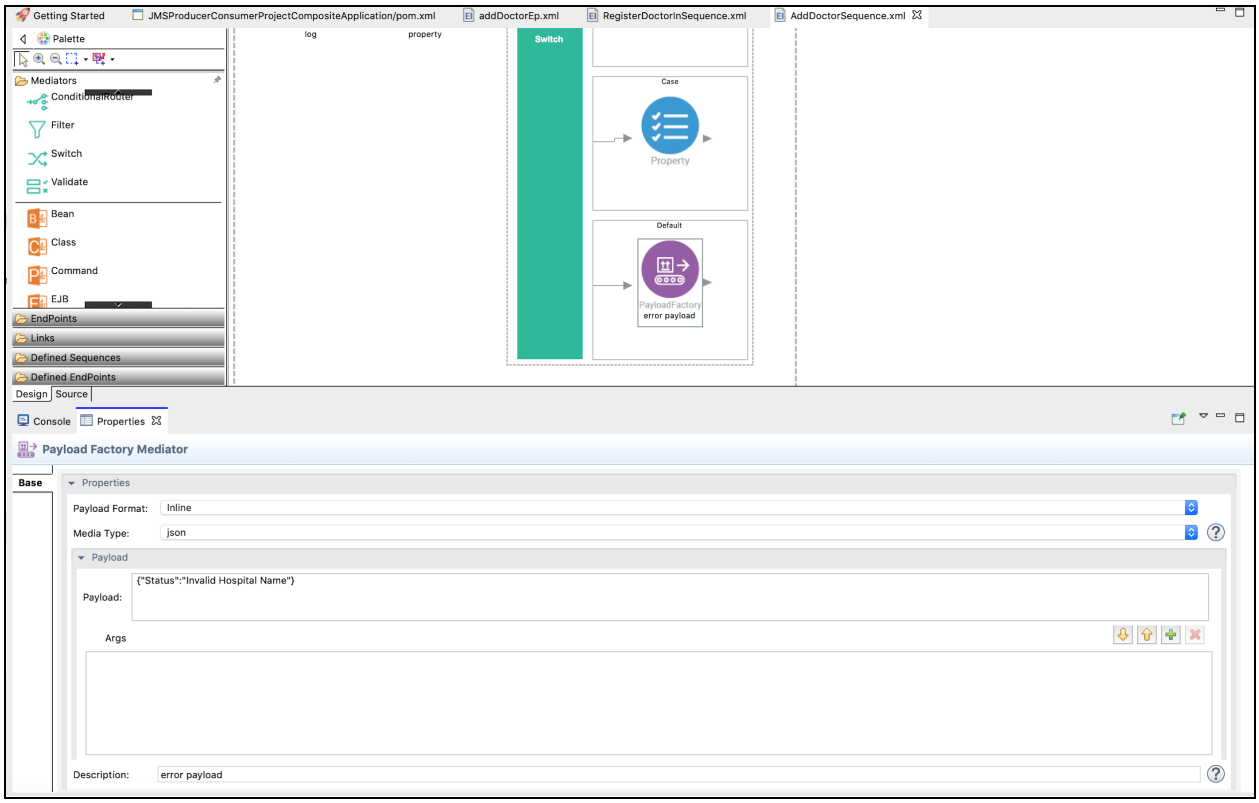
9. Add **Property** mediators to the other three cases and change the properties as follows:

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	willowgardens

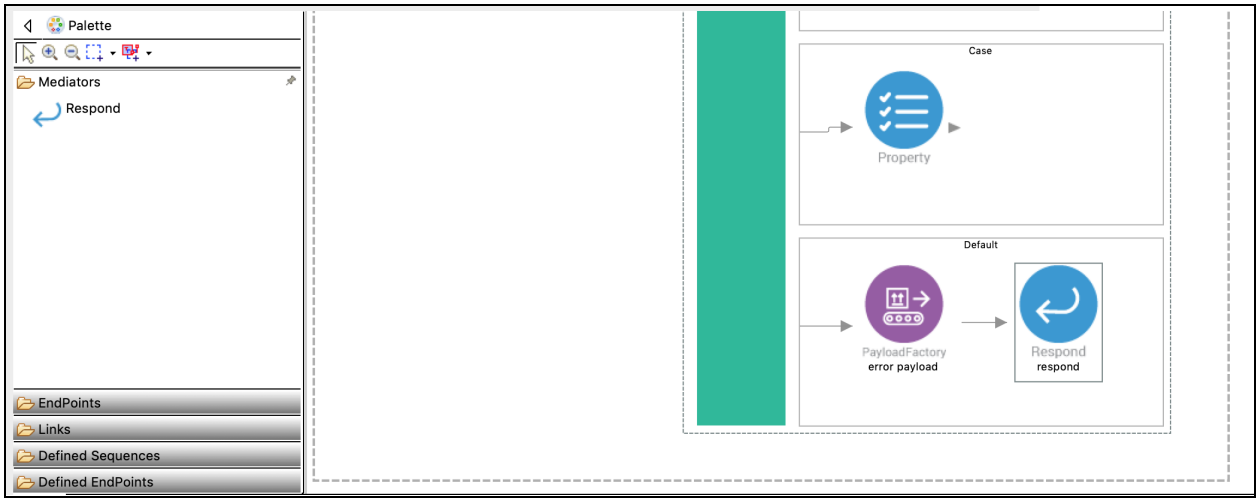


10. Drag and drop a **Payload Factory** mediator to the **Default** cell of the **Switch** mediator and change its properties as follows.

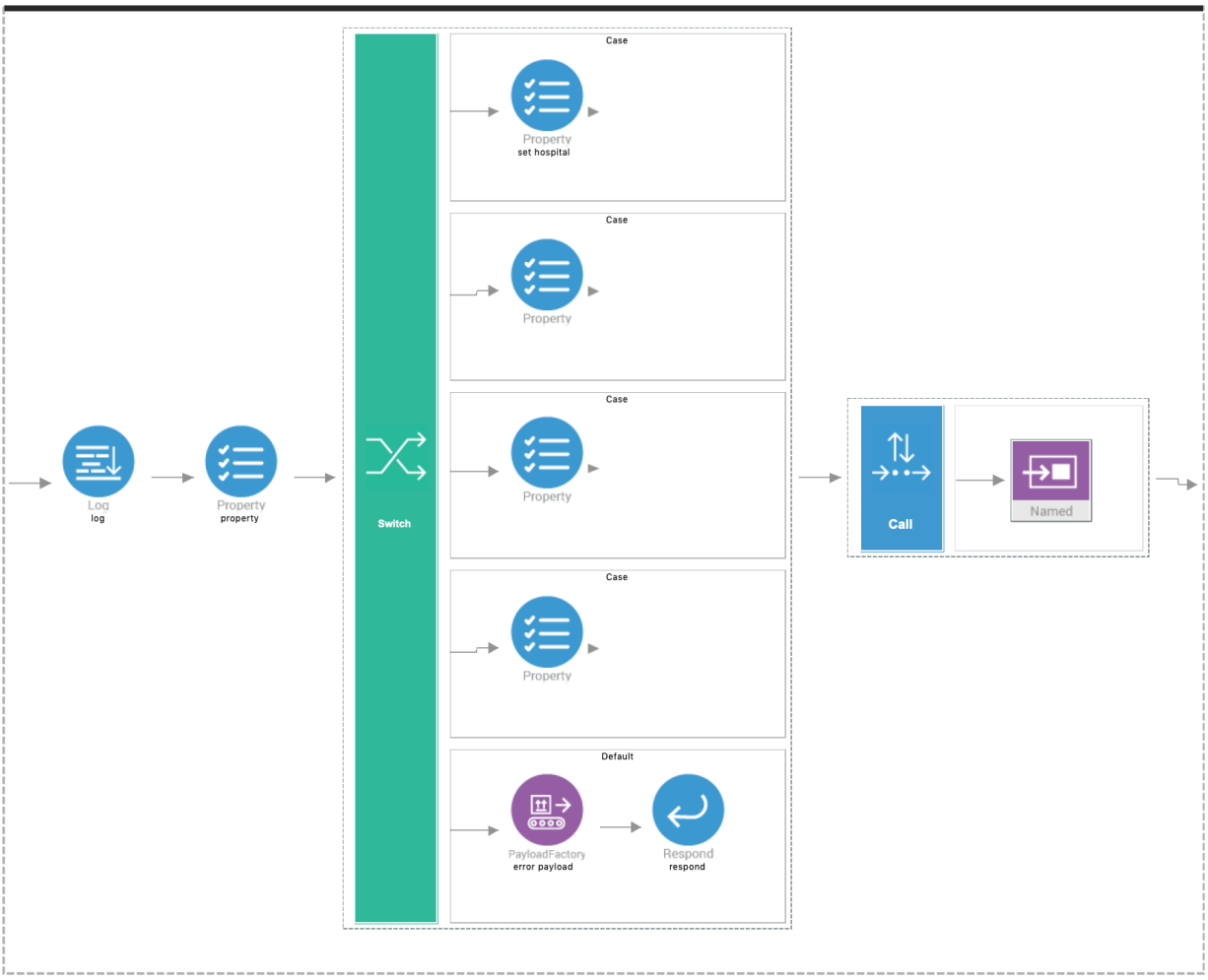
Property	**Value**
Payload Format	Inline
Media Type	json
Payload	{"Status":"Invalid Hospital Name"}



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

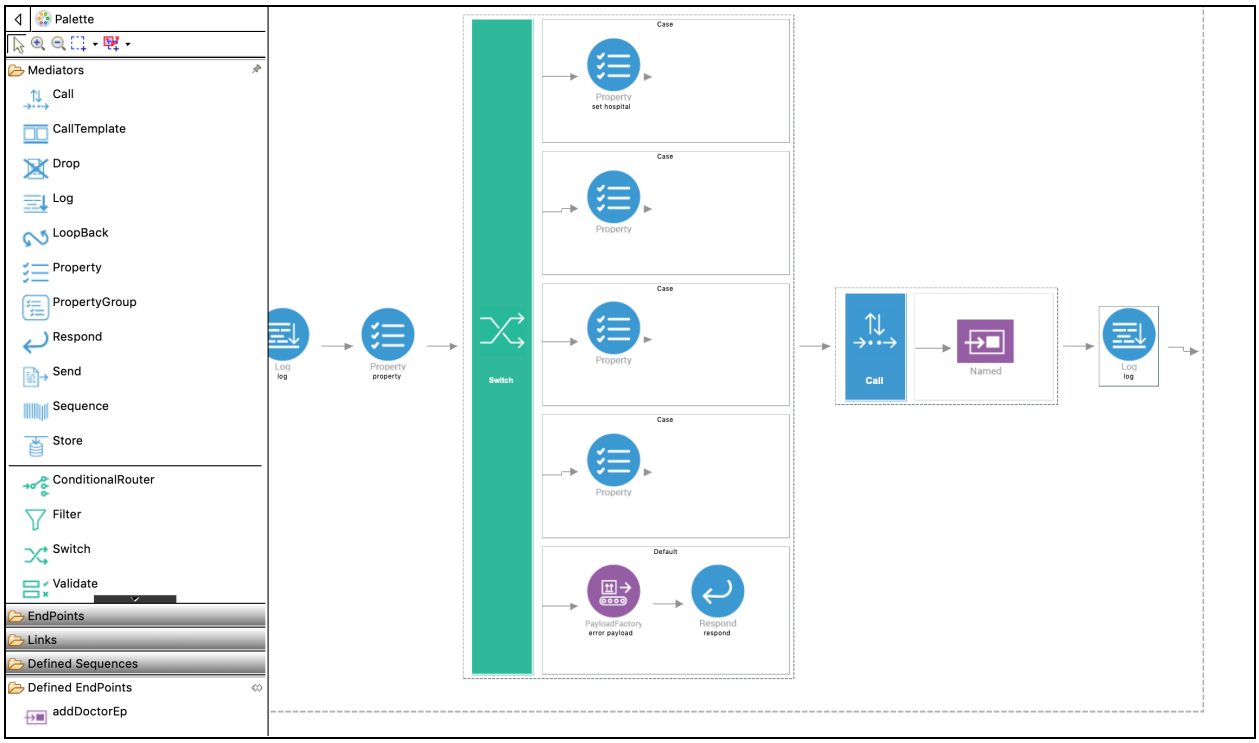


12. Drag and drop a **Call** mediator to the `addDoctorSequence` sequence and drag and drop the addDoctorEp endpoint from the **Defined Endpoints** panel of the **Palette** to the empty cell adjoining the **Call** mediator.



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

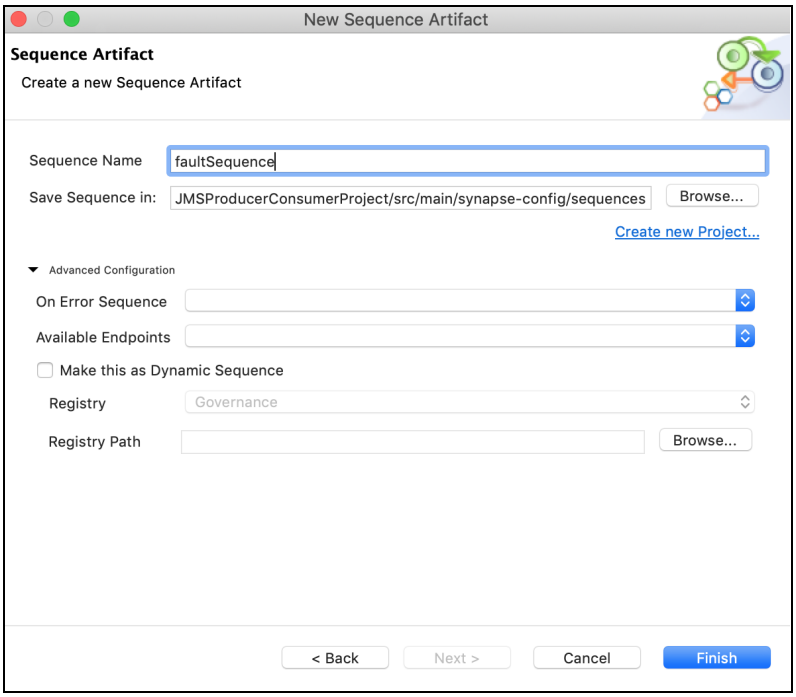
Property	Value
Log Category	INFO
Log Level	FULL



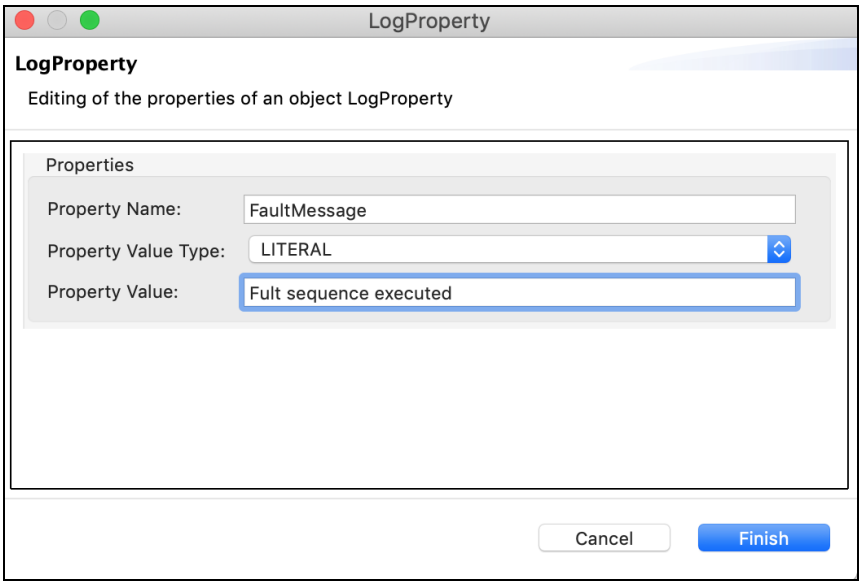
Creating sequence: `FaultSequence`

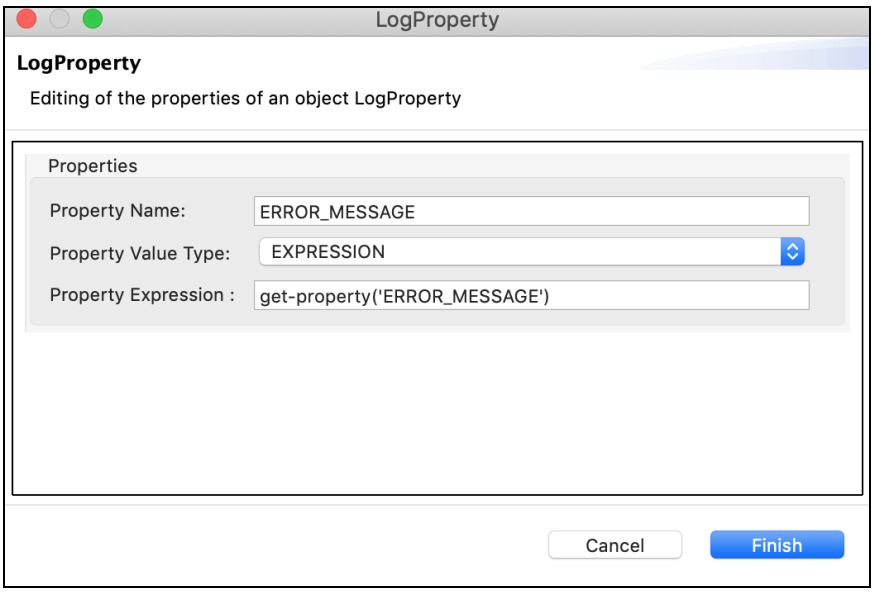
This sequence will be used to send the doctor details when an error occurs in the mediation.

1. Right-click the ESB Config module and select **New** -> **Sequence** to create a new sequence named `faultSequence`.



2. Drag and drop a **Log** mediator to the `faultSequence` sequence and change the following properties:

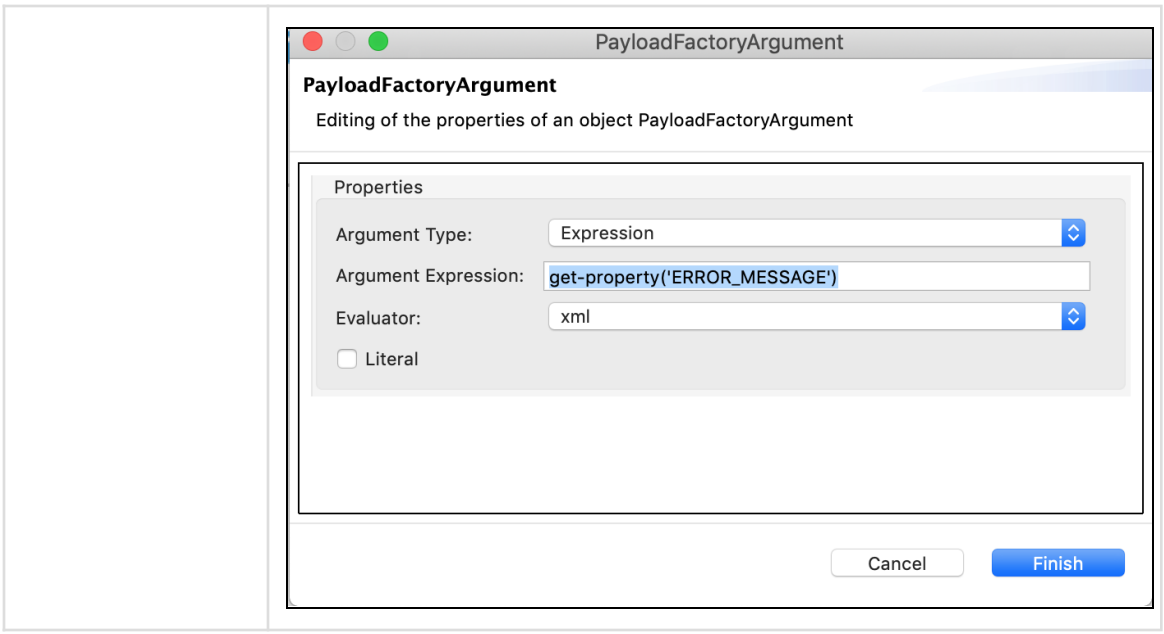
Property	Value
Log Category	INFO
Log Level	CUSTOM
Properties	Click the '+' icon to open the LogProperty dialog box and enter the following property values: **Name:** FaultMessage **Value/Expression:** Fault Sequence Executed  **Name:** ERROR_MESSAGE **Type:** EXPRESSION **Value/Expression:** Click on 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



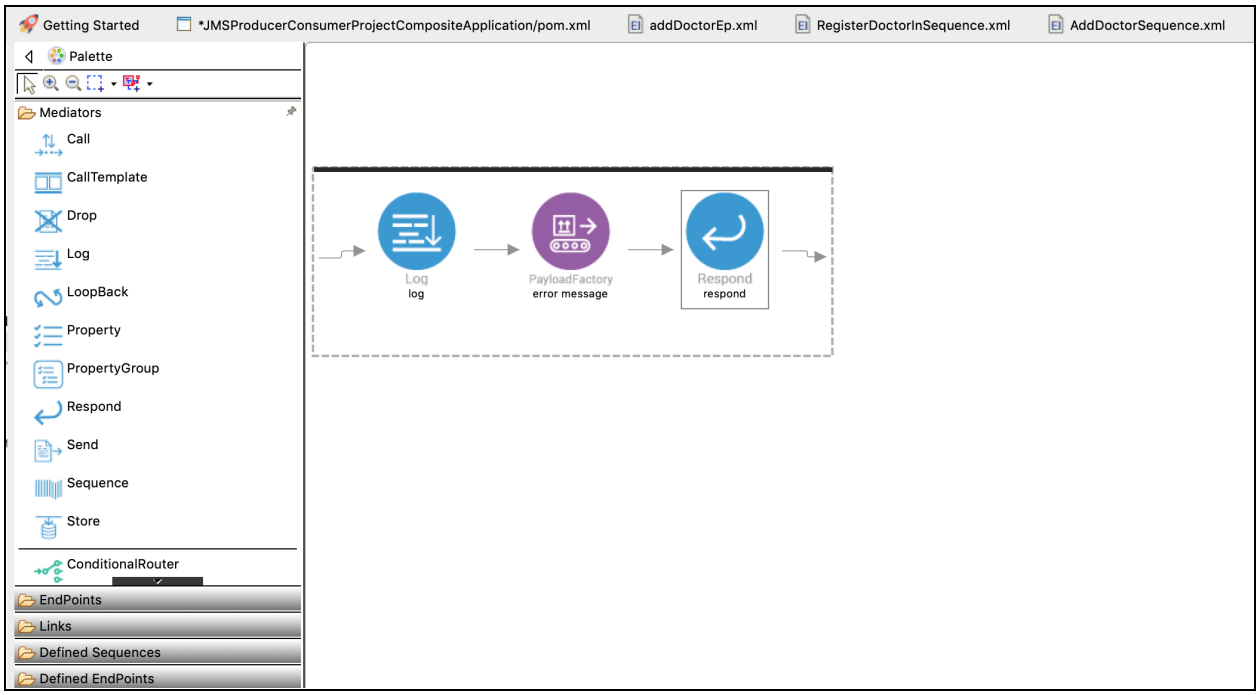
Click **Add** and click **OK**.

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

Property	**Value**
Payload Format	Inline
Media Type	json
Payload	`{"Status":"Error Occurred While Processing the Request",` `"Message":"$1"}`
Args	Click the '+' icon to enter arguments: • **Type:** Expression • **Value:** get-property('ERROR_MESSAGE') • **Evaluator:** xml

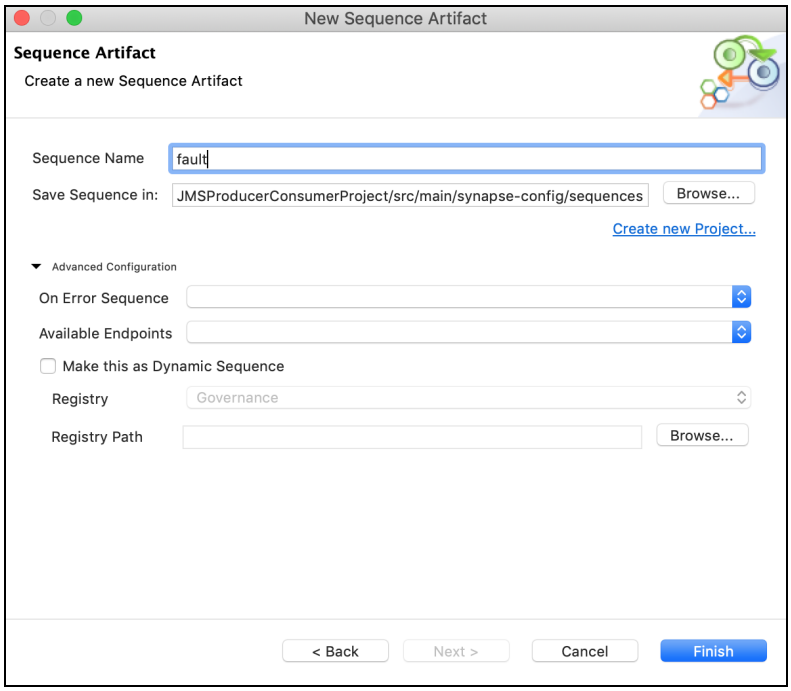


4. Drag and drop a **Respond** mediator to the `faultSequence` sequence.



Creating sequence: `fault`

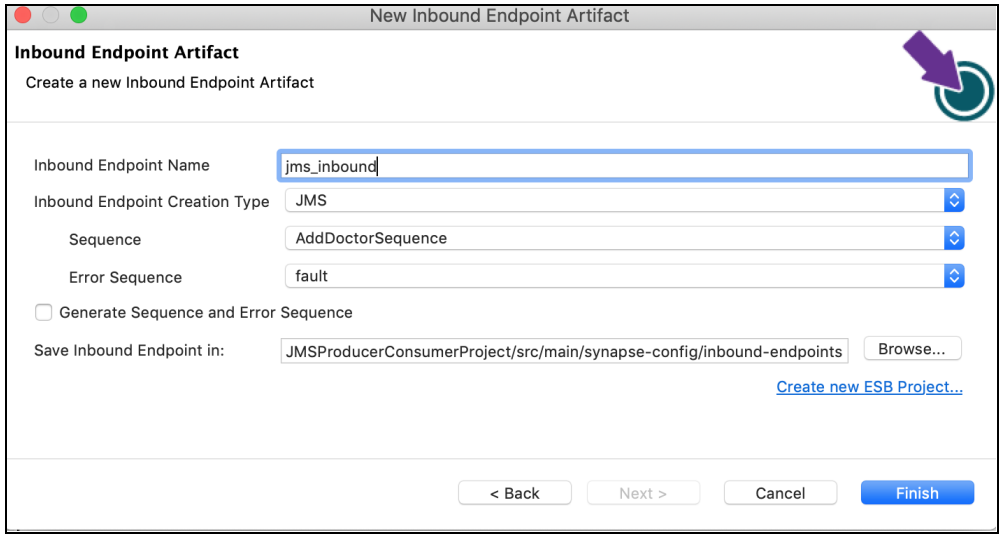
Right-click the project and select **New** -> **Sequence** to create a new sequence named `fault`.



Creating the Inbound Endpoint

The inbound endpoint will be used to listen to the JMS queue and consume messages. The messages are then passed to a sequence for processing.

1. Right-click the ESB Config module and select **New** -> **Inbound Endpoint** to create a new inbound endpoint named `jms_inbound`.
2. 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.
3. Select `AddDoctorSequence` as the **Sequence** and `fault` as the **Error Sequence**.



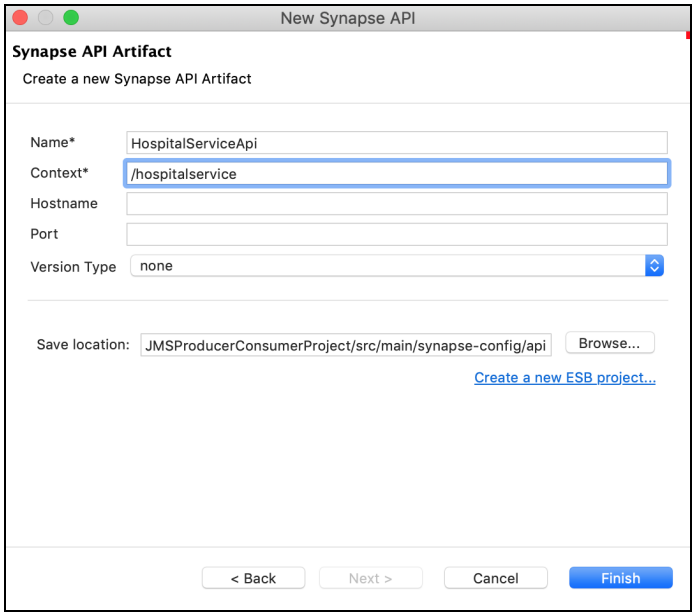
4. Double-click the inbound endpoint and change the following properties:

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

Creating the API

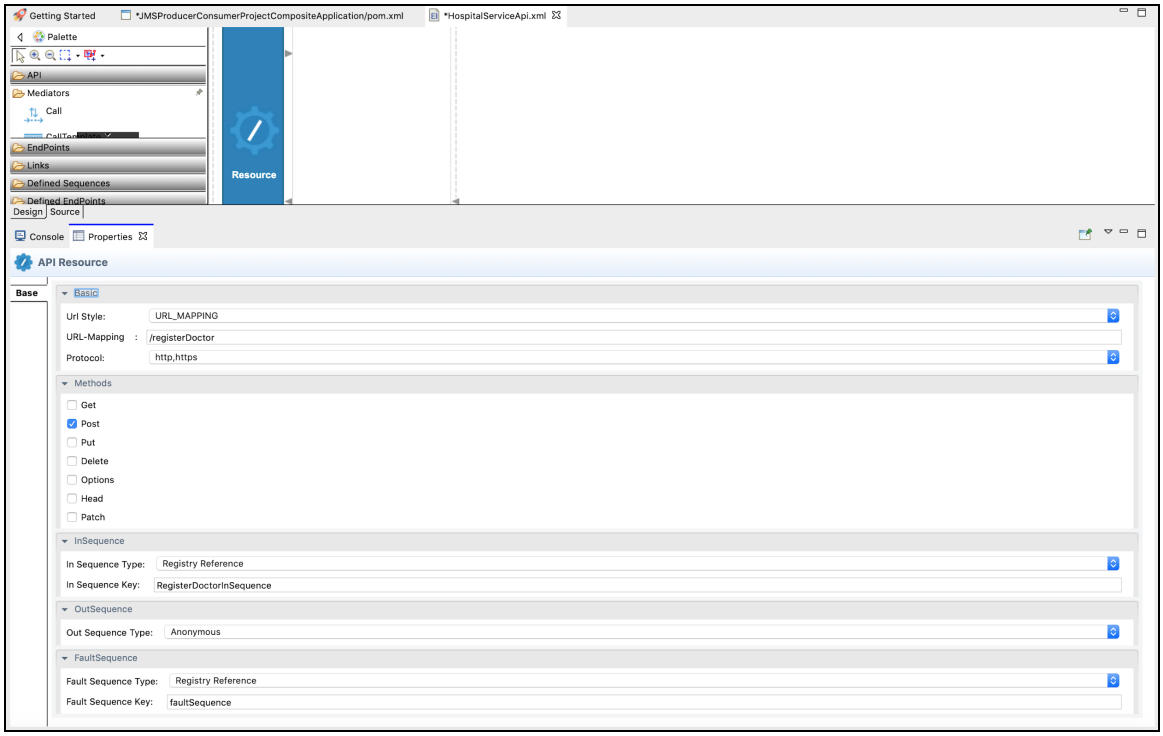
The REST API will receive messages from the HTTP client and then pass the messages to a sequence for processing.

1. Right-click the ESB Config module 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-Mapping	/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.

Creating the Composite Application

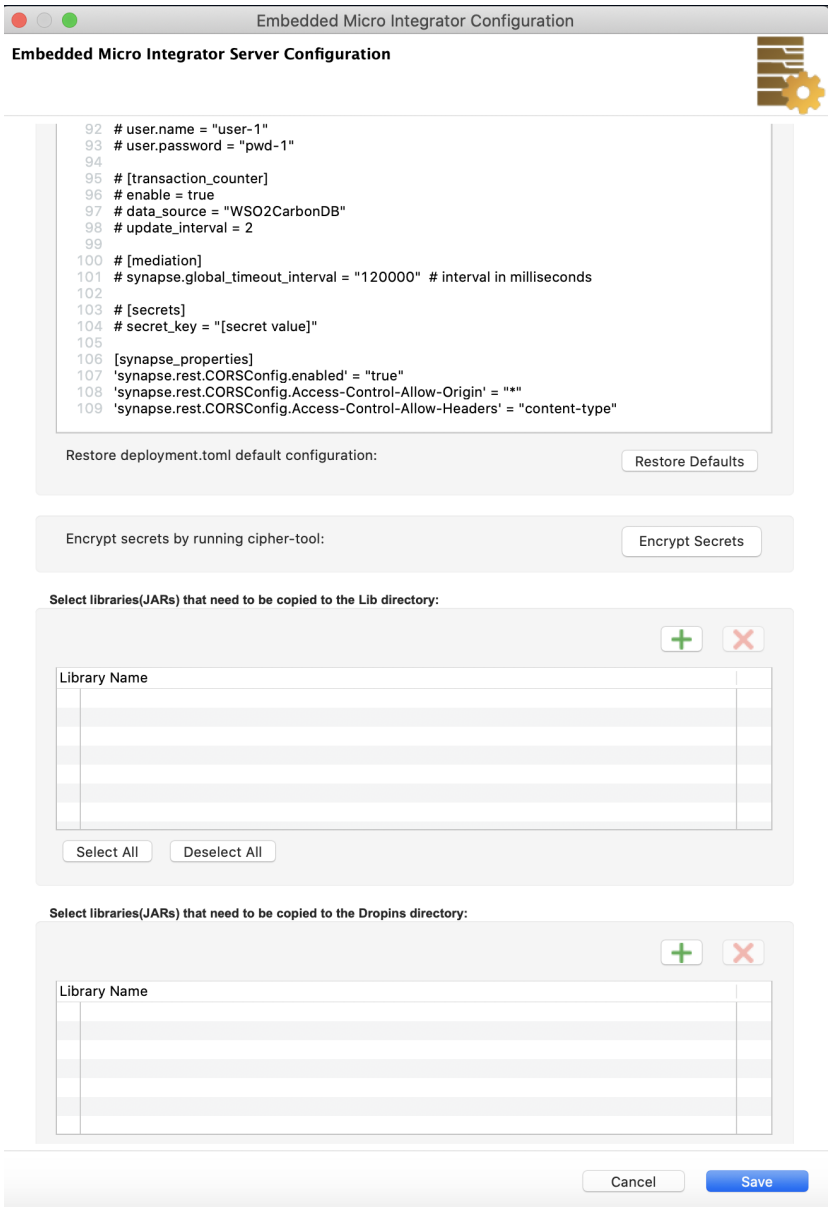
A composite exporter named `JMSProducerConsumerCompositeApplication` is already created in the Integration project.

Testing the Scenario

Configure Apache ActiveMQ with WSO2 Micro Integrator

- Download Apache ActiveMQ.

- Open the **Embedded Micro Integrator Configuration** panel in WSO2 Integration Studio to configure the embedded Micro Integrator with ActiveMQ.



- Copy the following client libraries from the `ACTIVEMQ_HOME/lib` directory to the `MI_HOME/lib` directory.

ActiveMQ 5.8.0 and above	Earlier version of ActiveMQ
• activemq-broker-5.8.0.jar • activemq-client-5.8.0.jar	• activemq-core-5.5.1.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)	• geronimo-j2ee-management_1.0_spec-1.0.jar • geronimo-jms_1.1_spec-1.1.1.jar

- Add the following configurations to enable the JMS sender with ActiveMQ connection parameters.

```
[[transport.jms.sender]]
name = "myQueueSender"
parameter.initial_naming_factory =
"org.apache.activemq.jndi.ActiveMQInitialContextFactory"
parameter.provider_url = "tcp://localhost:61616"
parameter.connection_factory_name = "QueueConnectionFactory"
parameter.connection_factory_type = "queue"
parameter.cache_level = "producer"
```

Start ActiveMQ

1. Open a terminal and navigate to the `ACTIVEMQ_HOME/bin` directory.
2. Execute `./activemq console` (on Linux/OSX) or `activemq start` (on Windows).

Start the back-end service

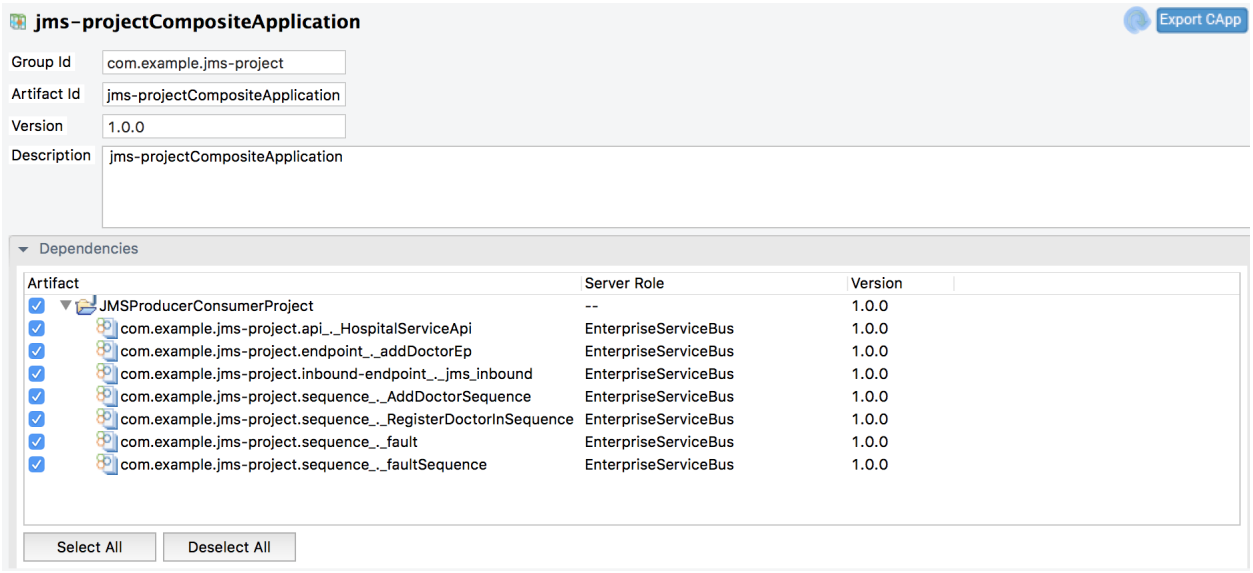
1. Download the JAR file of the Healthcare service from here.
2. Open a terminal, navigate to the location where you saved the JAR file.
3. Execute the following command to start the service:

```
java -jar Hospital-Service-JDK11-2.0.0.jar
```

Deploy the artifacts in the Micro Integrator

Let's deploy the artifacts in the embedded Micro Integrator of WSO2 Integration Studio.

1. Open the POM file of the composite application project and ensure that all the artifacts are added as dependencies:



2. Right-click the composite application project and click **Export Project Artifacts and Run**.

The Micro Integrator will start with the artifacts deployed.

Invoke the artifacts

When the artifacts are successfully deployed, you can test the scenario as follows:

1. 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"
}
```

2. In a new tab on 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`:

   Note that we are using cURL to simulate the HTTP client sending messages.

```
curl -v POST
"http://localhost:8290/hospitalservice/registerDoctor"
--header "Content-Type: application/json" -d @add_doctor.json
-k -v
```

3. 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 8290 (#0)
*   Trying ::1...
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 8290 (#0)
> POST /hospitalservice/registerDoctor HTTP/1.1
> User-Agent: curl/7.30.0
> Host: localhost:8290
> 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
```

4. View the response in Micro Integrator console (in WSO2 Integration Studio):

```
[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"}
```

5. 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"}
```