Exercise 8

Install the WSO2 ESB (and a backend service to invoke)
Using the ESB tooling inside Eclipse create a simple REST to SOAP mapping
Deploy a more complex ESB flow that implements the Jim Webber Starbucks REST model

Prior Knowledge
Working with SOAP and REST services
XML

Objectives
See basic ESB operations. Look at the mapping of REST to SOAP

Software Requirements
(see separate document for installation of these)

- Java Development Kit 7
- Eclipse
- WSO2 Developer Studio 2.1.0
- WSO2 Application Server 5.0.1
- WSO2 ESB 4.5.1

1. Before we install the ESB we need to host some services to interact with. To do this we are going to install the WSO2 Application Server. Unzip the wso2as-5.0.1.zip file:

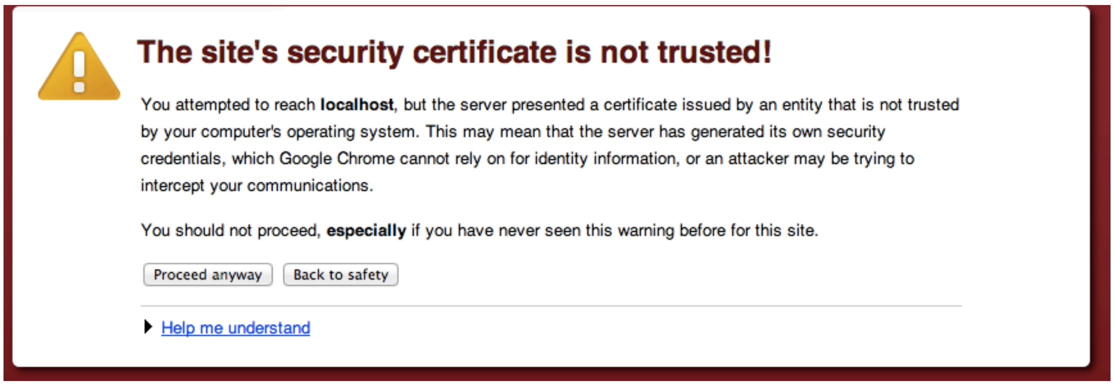
   ```
   cd ~/oxsoa
   unzip ~/Desktop/wso2as-5.0.1.zip
   ```

2. Make sure the scripts are executable:
   ```
   cd wso2as-4.5.1
   chmod +x bin/*.sh
   ```

3. Start up the server:
   ```
   bin/wso2server.sh
   ```

4. In Chrome, browse to
   https://localhost:9443 [Note the SSL URL]

   You will get a message like this:



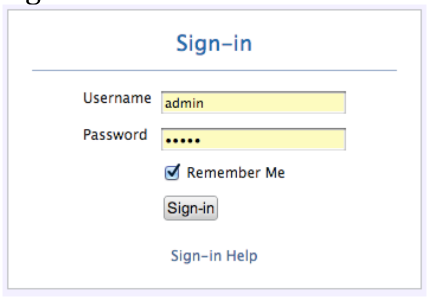
The site's security certificate is not trusted!

You attempted to reach **localhost**, but the server presented a certificate issued by an entity that is not trusted by your computer's operating system. This may mean that the server has generated its own security credentials, which Google Chrome cannot rely on for identity information, or an attacker may be trying to intercept your communications.

You should not proceed, **especially** if you have never seen this warning before for this site.

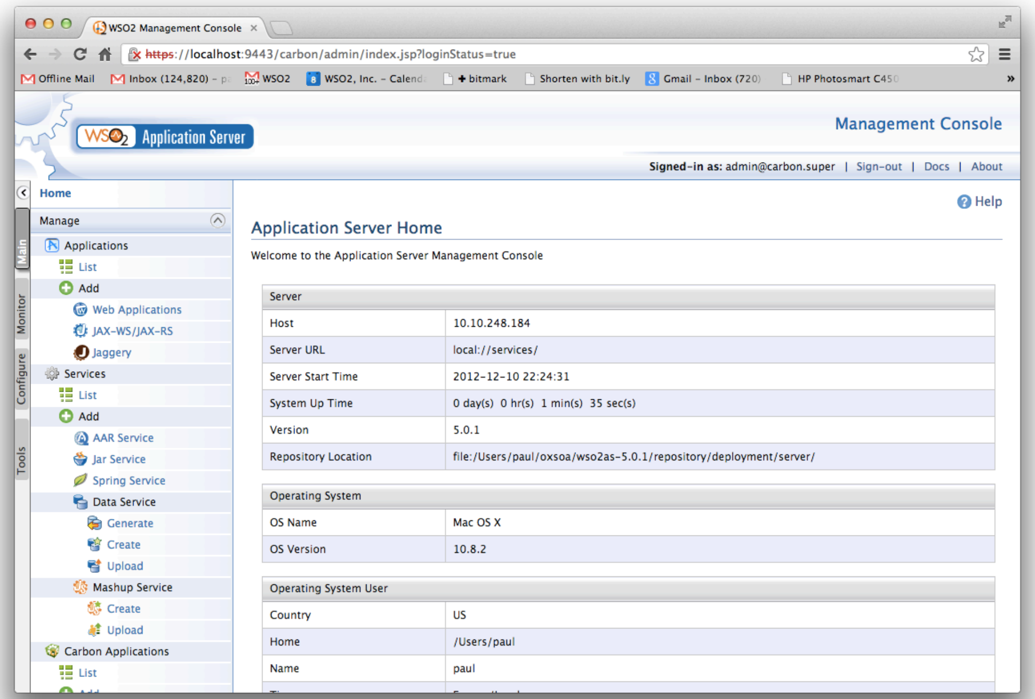
This is because we haven't installed a "proper" certificate into the server. Click **Proceed Anyway**

5. Use **admin/admin** as username and password. Click **Remember Me** and sign in:



Username	admin
Password	•••••

6. You should see a Web Console like this:



https://localhost:9443/carbon/admin/index.jsp?loginStatus=true

Application Server Home

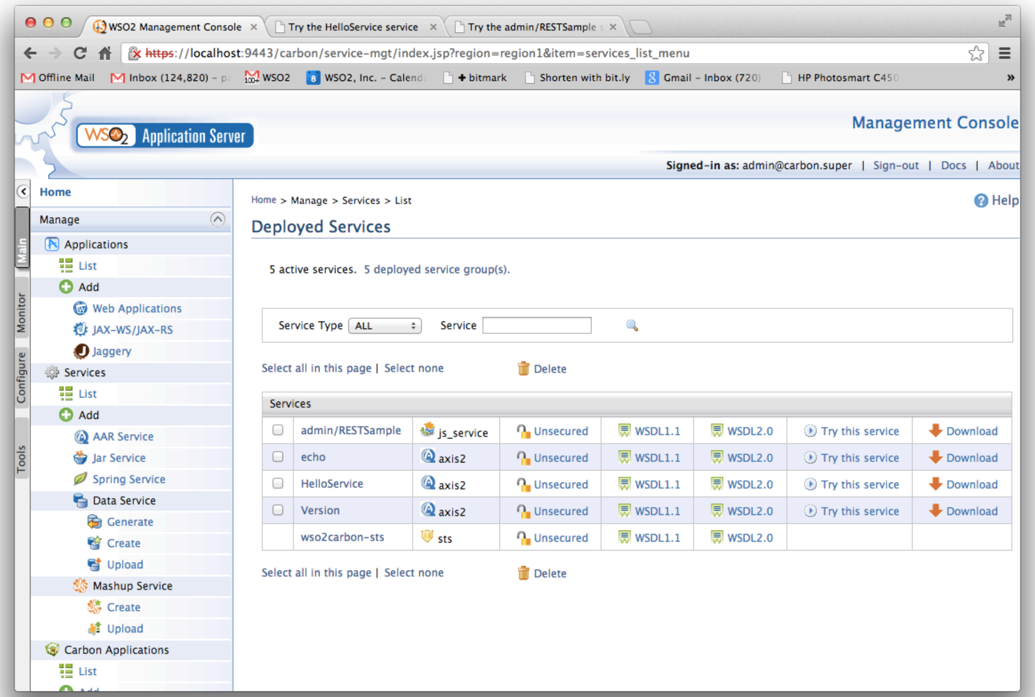
Welcome to the Application Server Management Console

Server	
Host	10.10.248.184
Server URL	local://services/
Server Start Time	2012-12-10 22:24:31
System Up Time	0 day(s) 0 hr(s) 1 min(s) 35 sec(s)
Version	5.0.1
Repository Location	file:/Users/paul/oxsoa/wso2as-5.0.1/repository/deployment/server/

Operating System	
OS Name	Mac OS X
OS Version	10.8.2

Operating System User	
Country	US
Home	/Users/paul
Name	paul

7. Click on **Services->List** in the left hand menu. You should see something like this:



Deployed Services

5 active services. 5 deployed service group(s).

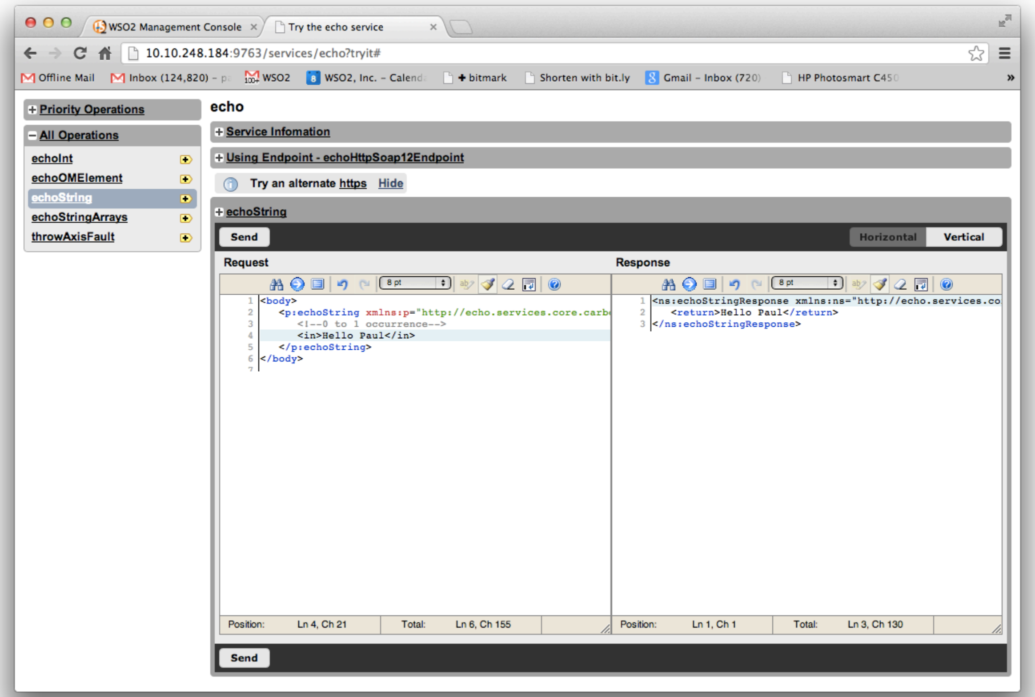
Service Type ALL Service

Select all in this page | Select none Delete

Services							
admin/RESTSample	js_service	Unsecured	WSDL1.1	WSDL2.0	Try this service	Download
echo	axis2	Unsecured	WSDL1.1	WSDL2.0	Try this service	Download
HelloService	axis2	Unsecured	WSDL1.1	WSDL2.0	Try this service	Download
Version	axis2	Unsecured	WSDL1.1	WSDL2.0	Try this service	Download
wso2carbon-sts	sts	Unsecured	WSDL1.1	WSDL2.0		

Select all in this page | Select none Delete

8. To see if the basic "echo" service is working click on "Try this service" next to echo.
9. You will see a "test" client.



Request:

```
<body>
   <p:echoString xmlns:p="http://echo.services.core.carb
      <!--0 to 1 occurrence-->
      <in>Hello Paul</in>
   </p:echoString>
</body>
```

Position: Ln 4, Ch 21 Total: Ln 6, Ch 155

Response:

```
<ns:echoStringResponse xmlns:ns="http://echo.services.co
   <return>Hello Paul</return>
</ns:echoStringResponse>
```

Position: Ln 1, Ch 1 Total: Ln 3, Ch 130

10. Select the **echoString** operation, modify the XML **(replace the ?)** and click **Send.**

11. Close that tab to get back to the main console.

12. Download the Starbucks Outlet Service:
    Browse to http://freo.me/UtN4Mj

13. Click on **Services/Add/AAR Service.**

14. Click on **Choose File**. Browse to the saved StarbucksOutletService.aar
    This is an Axis2 Web Service application. Click **Upload**

15. Wait a minute for this to deploy, then Click on **Services/List** again.

16. Now Click on **Try this service** for the StarbucksOutletService.

17. Create an order, list orders, pay for it, etc. Get a feel for the SOAP API.

18. Go to the Monitor tab and look at some stats, logs, etc.

19. **INSTALLING THE ESB**

20. Start a new command line window
21. Unzip and install the WSO2 ESB:

    ```
    cd ~/oxsoa
    unzip ~/Desktop/wso2esb-4.5.1.zip
    ```

22. Make sure the scripts are executable:
    ```
    cd wso2esb-4.5.1
    chmod +x bin/*.sh
    ```

23. Before starting the server, we have to cure a problem. The ESB and App Server both will want to bind to the same network ports, which is not allowed.

24. Luckily there is an easy fix. Edit the **~/oxsoa/wso2esb-4.5.1/repository/conf/carbon.xml** config file

    You can use nano on the command line or gedit in the windowing environment.

25. Search for offset and change the entry to read:

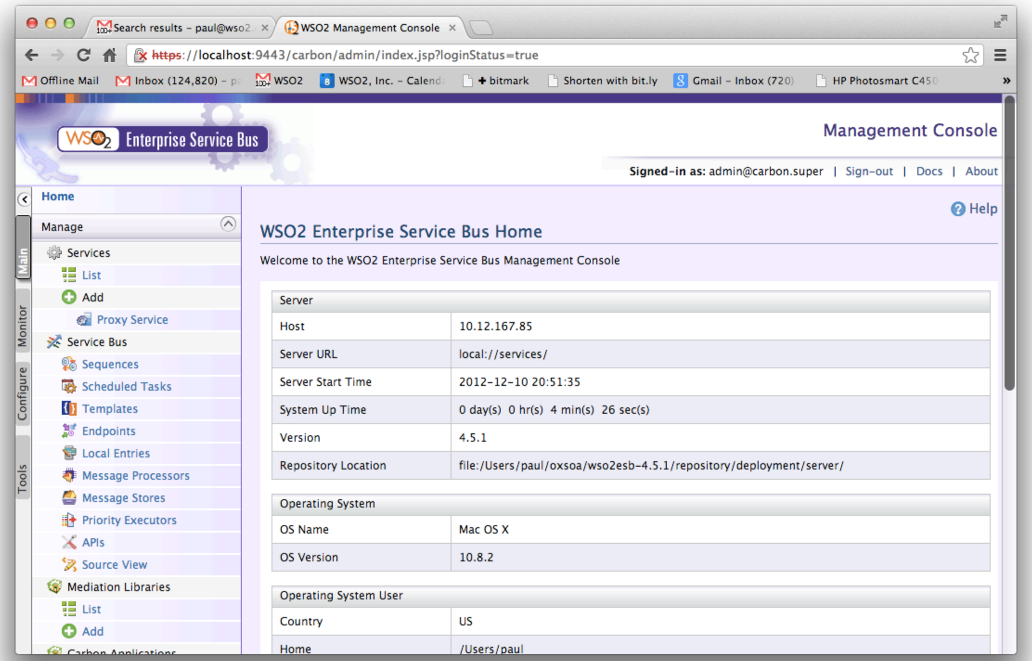
    ```
    <Offset>1</Offset>
    ```

26. This will move all the ports up by 1.

27. Start the server up
    ```
    bin/wso2server.sh
    ```

28. In Chrome, browse to https://localhost:9444 [Note the SSL URL]

29. Login as before. You should see a similar (but different) console. Compare to the App Server screens.



WSO2 Enterprise Service Bus Home

Welcome to the WSO2 Enterprise Service Bus Management Console

Server	
Host	10.12.167.85
Server URL	local://services/
Server Start Time	2012-12-10 20:51:35
System Up Time	0 day(s) 0 hr(s) 4 min(s) 26 sec(s)
Version	4.5.1
Repository Location	file:/Users/paul/oxsoa/wso2esb-4.5.1/repository/deployment/server/

Operating System	
OS Name	Mac OS X
OS Version	10.8.2

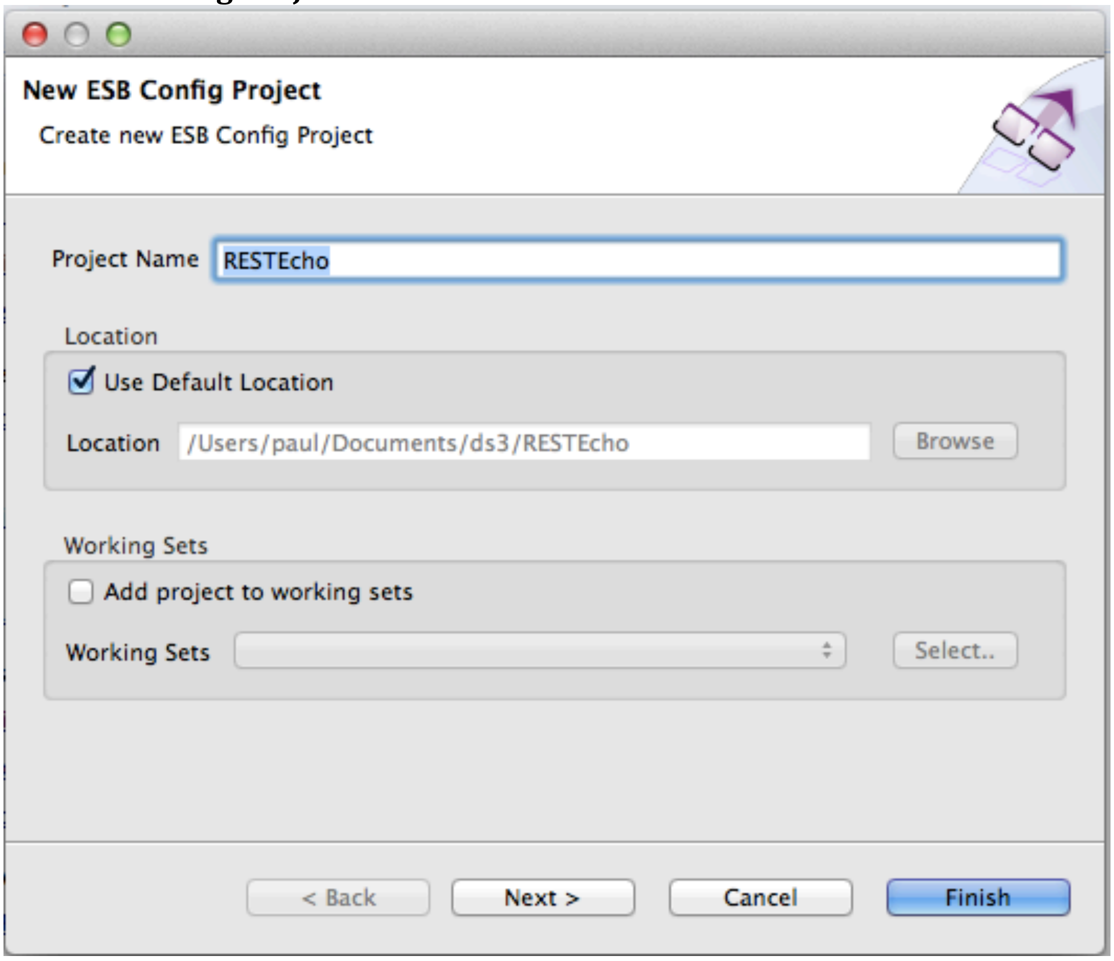
Operating System User	
Country	US
Home	/Users/paul

30. Now we are going to create a simple REST to SOAP mapping. We will initially use the echo service available in the AppServer.

31. Start or switch to Eclipse

32. Goto the Developer Studio Dashboard

33. Click **ESB Config Project**



New ESB Config Project

Create new ESB Config Project

Project Name: RESTEcho

Location

[✓] Use Default Location

Location: /Users/paul/Documents/ds3/RESTEcho

Working Sets

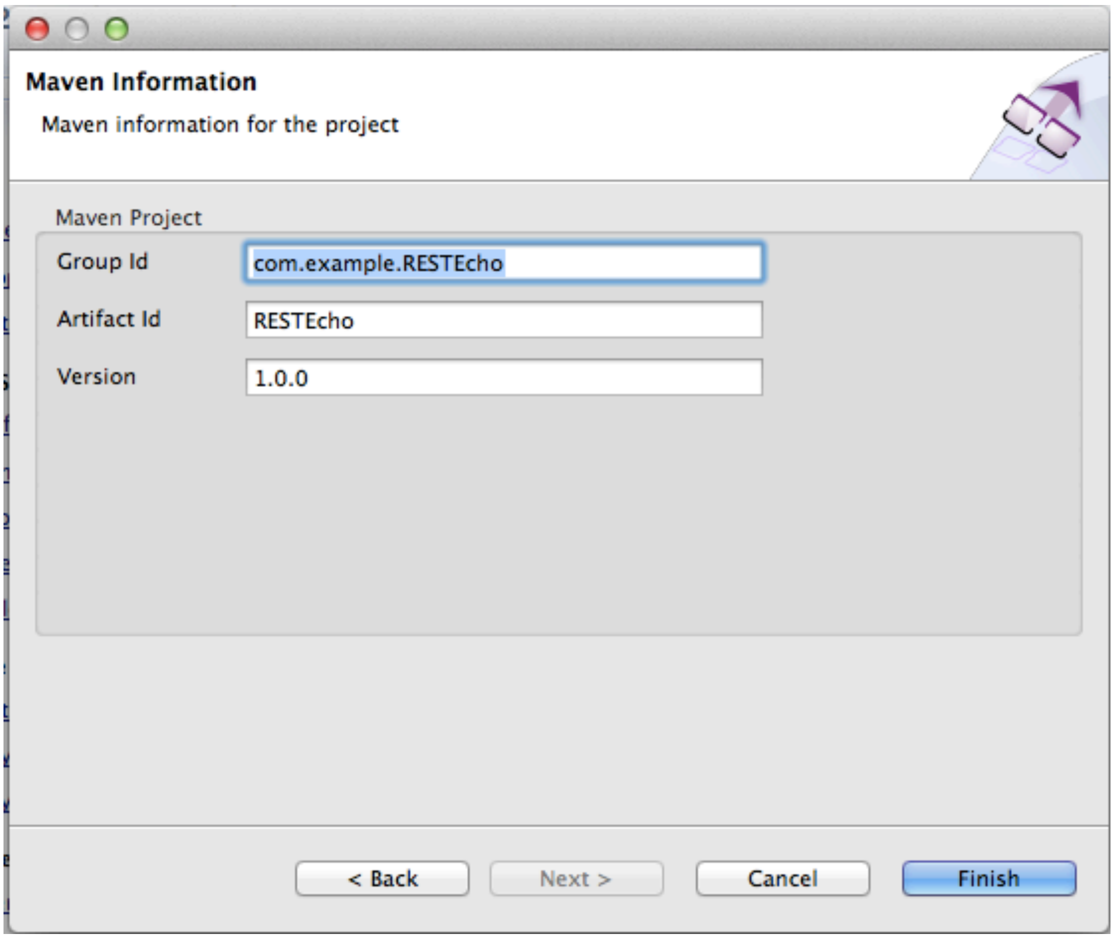
[ ] Add project to working sets

Working Sets:

34. Use the name **RESTEcho**

35. Click **Next**

36. Leave the Maven info the same:



Maven Information

Maven information for the project

Maven Project

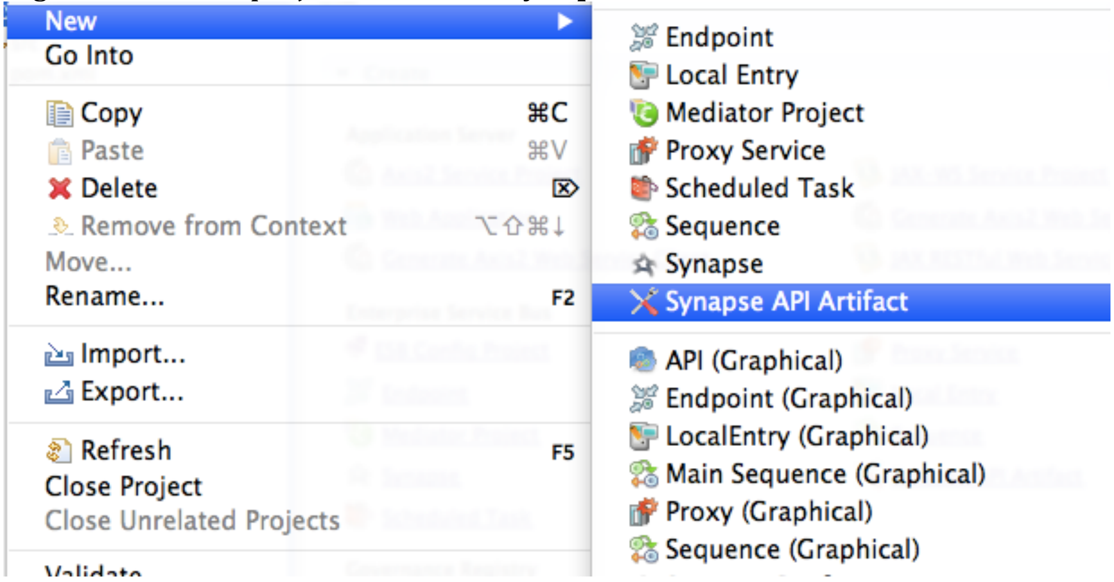
Group Id: com.example.RESTEcho

Artifact Id: RESTEcho

Version: 1.0.0

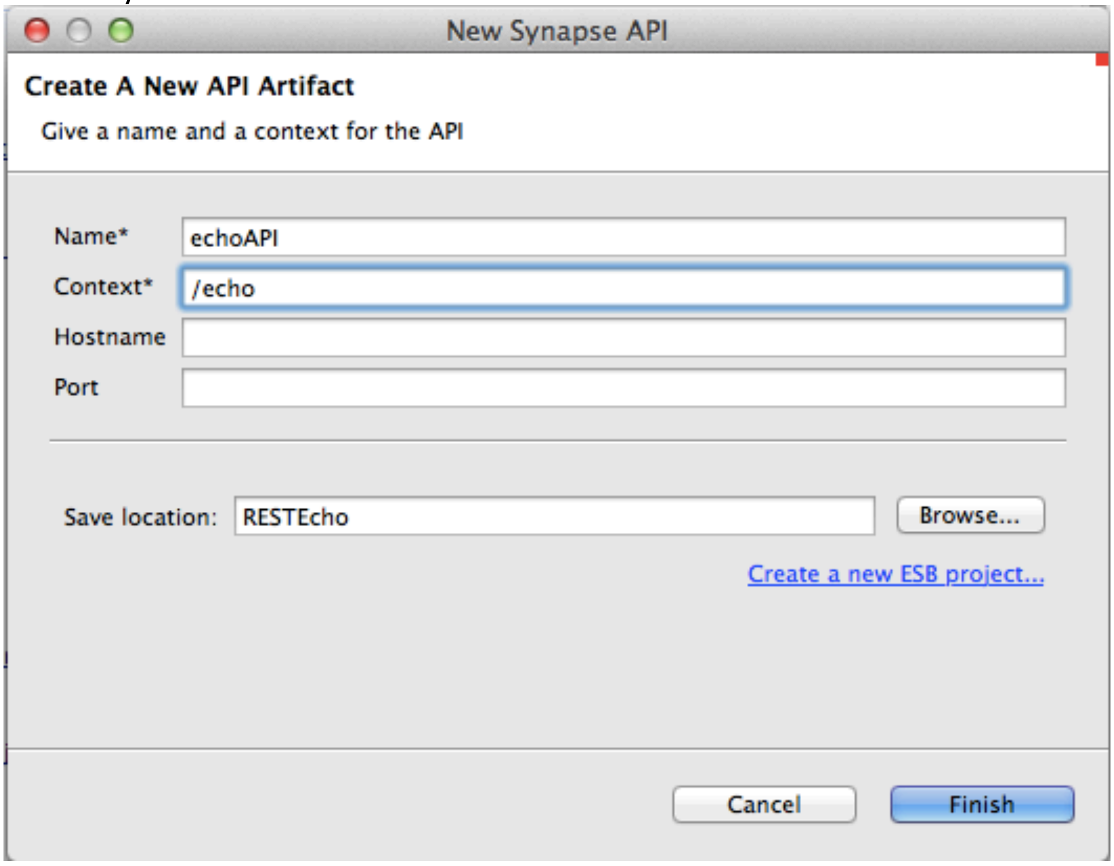
37. Click **Finish**

38. Right Click on the project and **New->Synapse API Artifact**



39. Use:

Name: **echoAPI**
Context: **/echo**


New Synapse API

Create A New API Artifact

Give a name and a context for the API

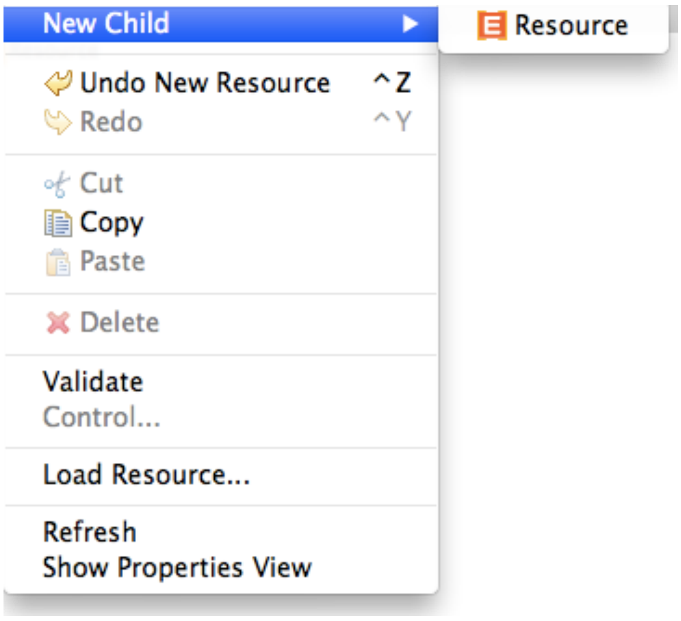
Name*: echoAPI
Context*: /echo
Hostname:
Port:

Save location: RESTEcho

40. Finish

41. Right click on the echoAPI item.

42. Choose New Child->Resource:

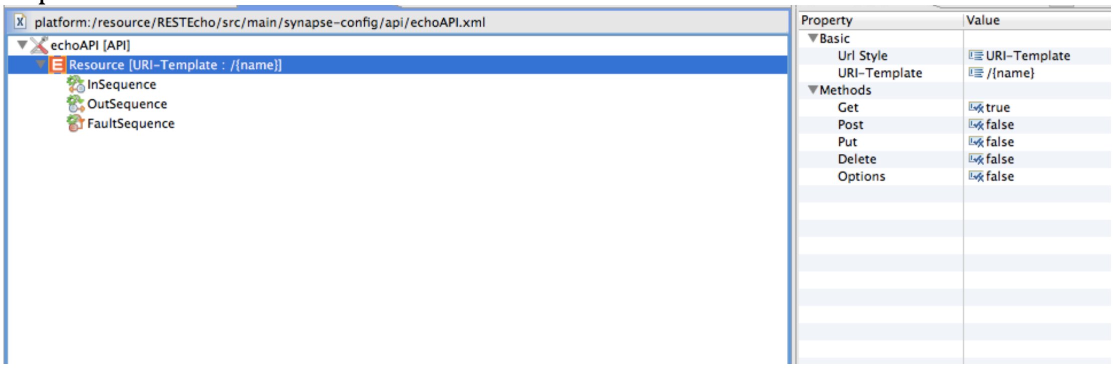


43. Select the Resource and on the right hand side edit the properties to:

URL Style: URI Template
URI-Template: /{name}
Get: true

Note the template option only appears after you select the style.

44. Expand the little arrow next to the Resource:

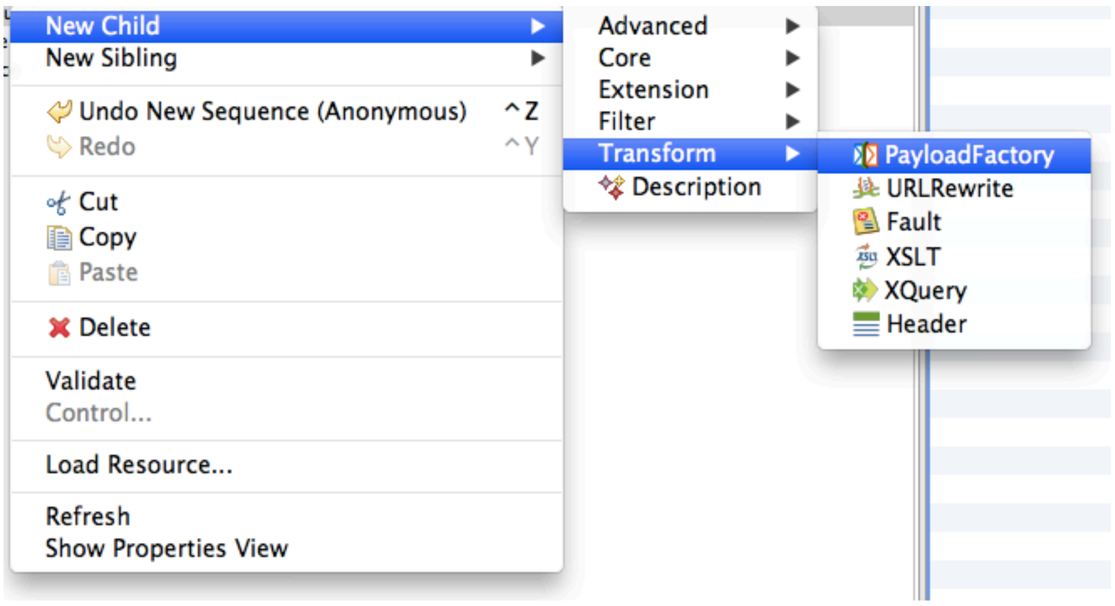


```
platform:/resource/RESTEcho/src/main/synapse-config/api/echoAPI.xml
▼ echoAPI [API]
  ▼ Resource [URI-Template : /{name}]
      InSequence
      OutSequence
      FaultSequence
```

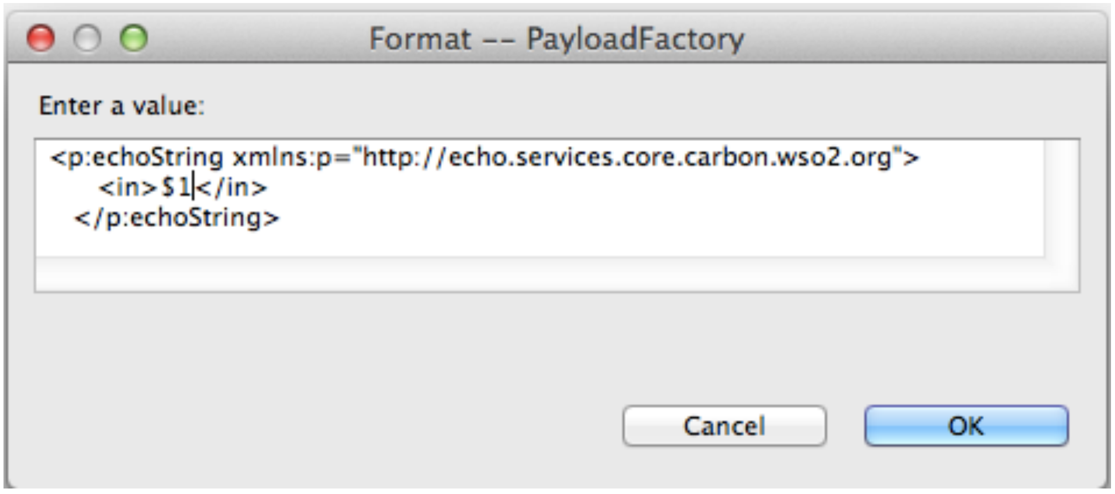
Property	Value
▼Basic	
Url Style	URI-Template
URI-Template	/{name}
▼Methods	
Get	true
Post	false
Put	false
Delete	false
Options	false

45. Right click on InSequence and New Child-> Sequence (Anonymous)

46. Right click on the anonymous sequence and add a
New Child->Transform->PayloadFactory:

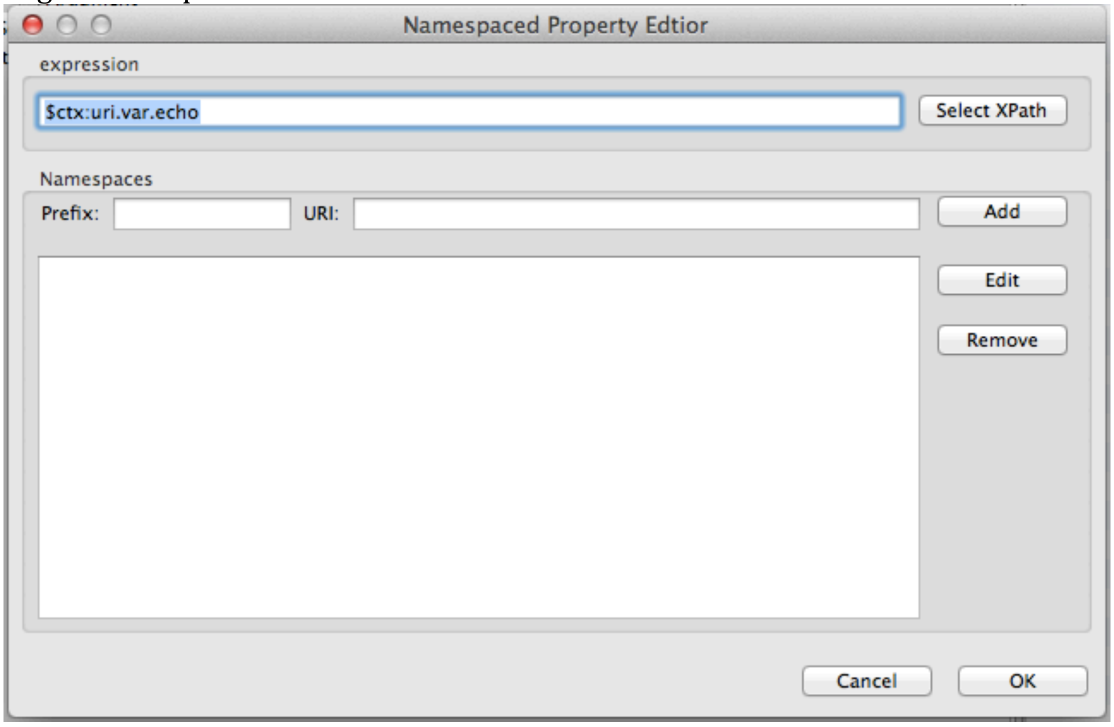


47. Click on the PayloadFactory mediator and using the properties pane on the right hand side, edit the "Format" replacing <inline/> with the XML below.



```
<p:echoString xmlns:p="http://echo.services.core.carbon.wso2.org">
    <in>$1</in>
  </p:echoString>
```

48. Add a new **Argument** child to the PayloadFactory. Use the following properties:
Argument Type: Expression
Argument Expression: $ctx:uri.var.name

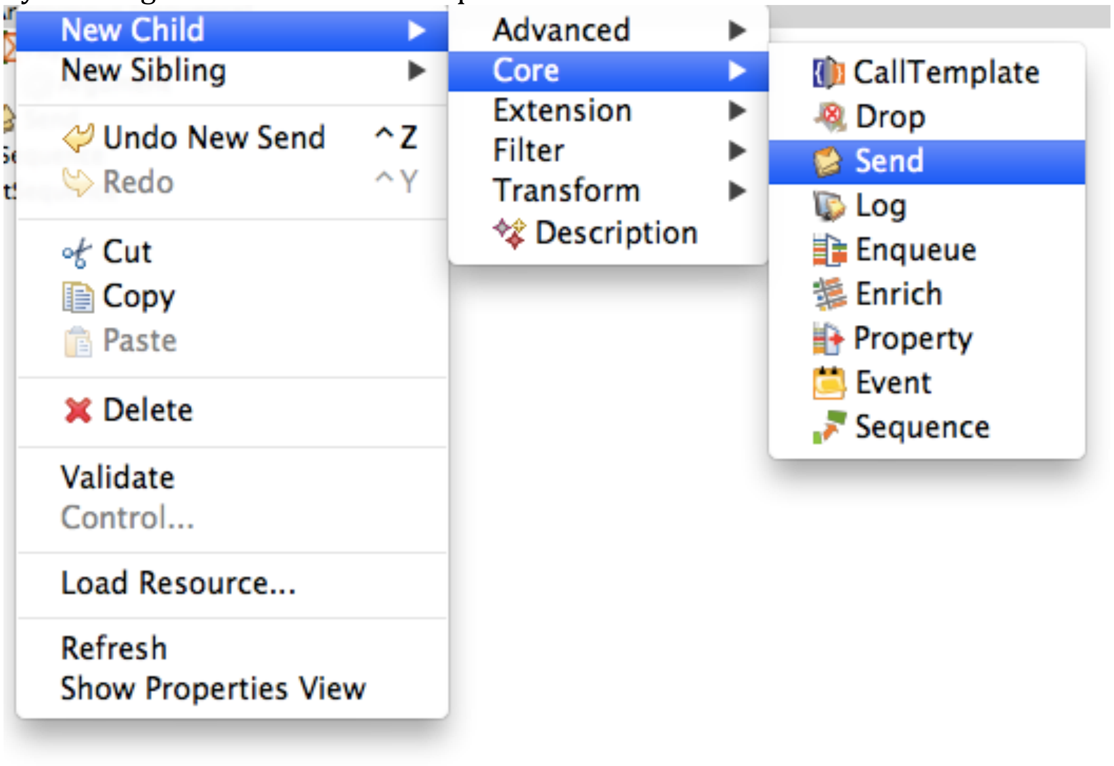


Namespaced Property Edtior

expression: $ctx:uri.var.echo

This will grab the {name} property from the incoming URL and map it to argument $1, which will be embedded in the XML.

49. Now add a new Send mediator below the PayloadFactory. You can do this by creating a new child on the sequence.



50. On a Send mediator create a new Child -> Address Endpoint

51. Change the properties on the Address Endpoint:
Address: http://127.0.0.1:9763/services/echo
Message Format: SOAP1.1

(This is the endpoint of the echo SOAP service we tested earlier)

52. You have now created an ESB API that will:
- Listen at /echo/{name}
- Extract the {name} value
- Construct an XML message
  - Using the name parameter
- Send this as SOAP11 to our server endpoint
- Send the response back to the client

53. Before deploying this in the ESB, take a look at the XML configuration behind this configuration. Click on the Source tab (bottom left corner of the API design pane). Your XML should look like this:

```
<api xmlns="http://ws.apache.org/ns/synapse" context="/echo"
name="echoAPI">
    <resource methods="GET" uri-template="/{name}">
        <inSequence>
            <payloadFactory>
                <format>
                    <p:echoString
xmlns:p="http://echo.services.core.carbon.wso2.org">
                        <in
xmlns="http://ws.apache.org/ns/synapse">$1</in>
                    </p:echoString>
                </format>
                <args>
                    <arg expression="$ctx:uri.var.name"/>
                </args>
            </payloadFactory>
            <send>
                <endpoint>
                    <address encoding="UTF-8" format="soap11"
                        statistics="disable" trace="disable"
uri="http://127.0.0.1:9763/services/echo">
                        <timeout>
                            <duration>0</duration>
                            <responseAction>discard</responseAction>
                        </timeout>
                        <markForSuspension>

<retriesBeforeSuspension>0</retriesBeforeSuspension>
                            <retryDelay>0</retryDelay>
                        </markForSuspension>
                        <suspendOnFailure>
                            <initialDuration>0</initialDuration>
                            <maximumDuration>0</maximumDuration>
                            <progressionFactor>1.0</progressionFactor>
                        </suspendOnFailure>
                    </address>
                </endpoint>
            </send>
        </inSequence>
    </resource>
</api>
```

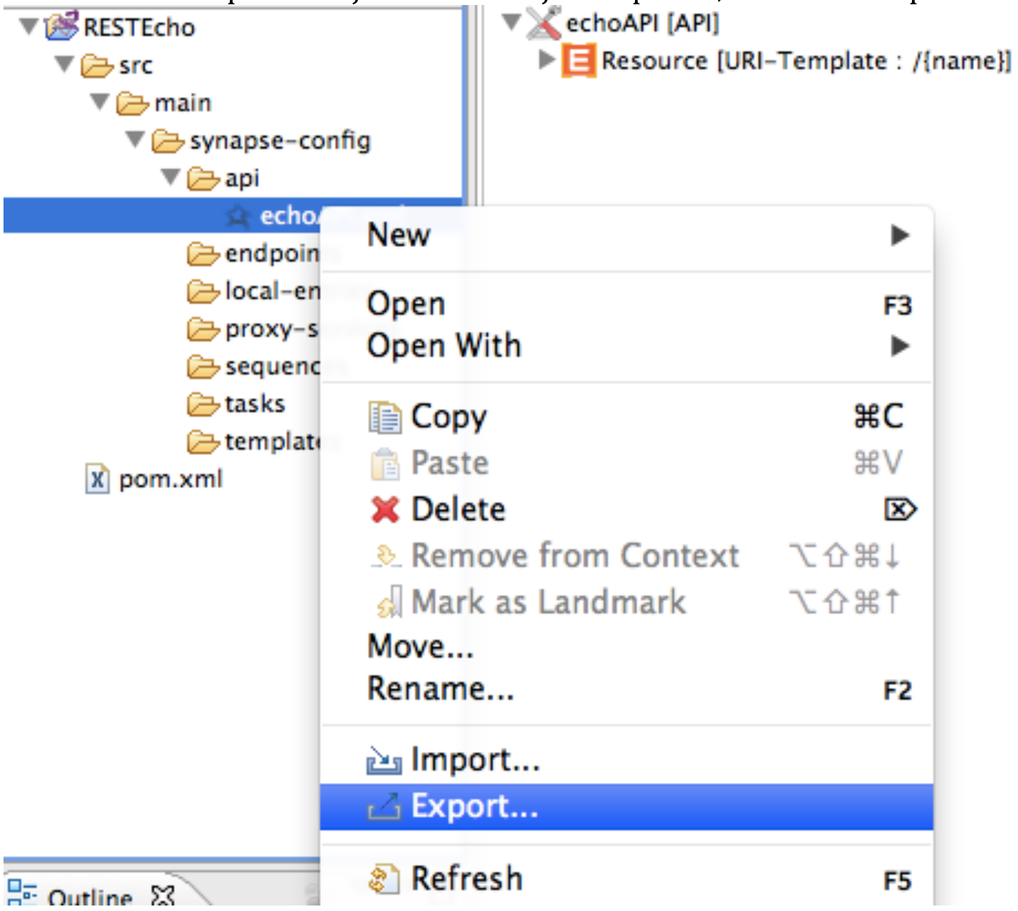
54. Let's briefly examine this:

Firstly, we are defining an API which is a collection of resource definitions (in the REST style). Each resource is actually implemented by a sequence of flow logic. In this case, we are looking for a GET and mapping it to a simple flow with two mediators. First we create an XML payload, then we send that to an endpoint. The endpoint definition is quite long because the editor has filled in plenty of config for what to do in failure cases.

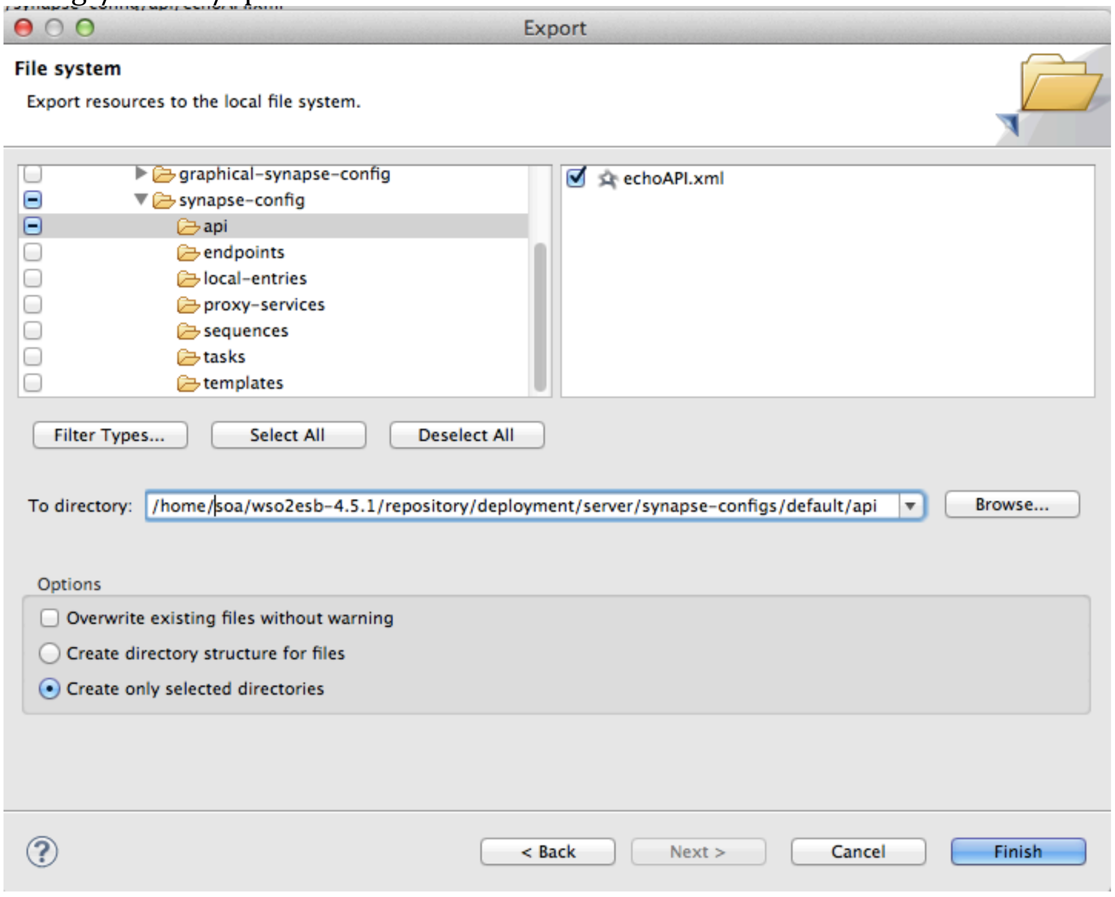
55. Now we need to deploy this into the ESB. Unfortunately there is a bit of a bug since this is an early build of Dev Studio 3.0.0.

56. Instead of doing the normal code deploy (which involves creating a Carbon Archive – CAR file) and deploying on the server, we are going to directly drop the api xml file into the ESB's deployment directory.

57. Select the echoApi.xml object in the Project Explorer, and select Export:



58. Choose **File System**
    For the **To Directory** location choose the following directory:
    /home/soa/wso2esb-4.5.1/repository/deployment/server/synapse-configs/default/api



To directory: /home/soa/wso2esb-4.5.1/repository/deployment/server/synapse-configs/default/api

Options
- [ ] Overwrite existing files without warning
- ○ Create directory structure for files
- ● Create only selected directories

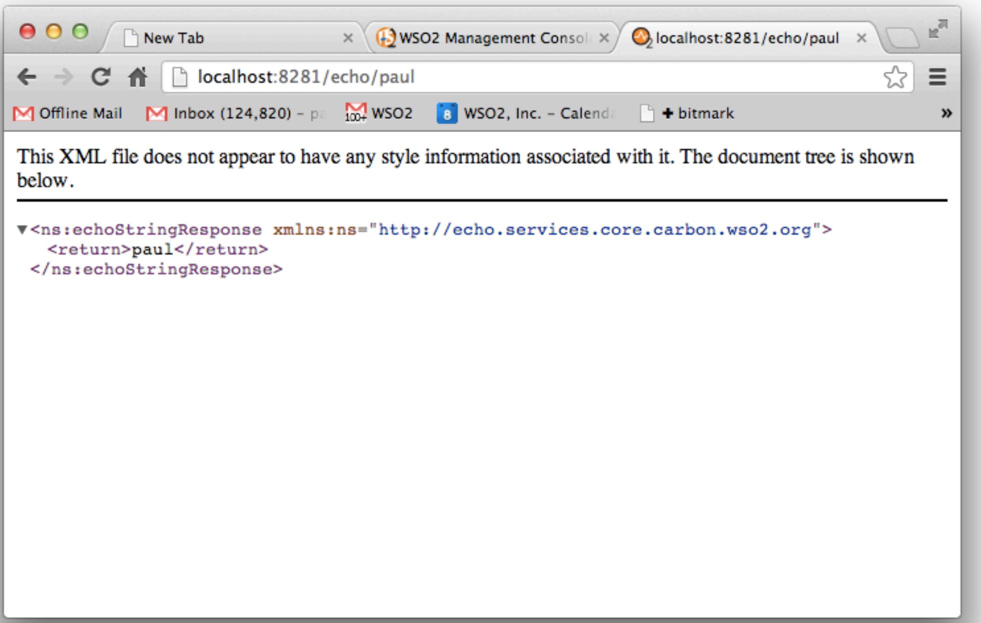
59. Click **Finish**

60. Go to the ESB command line window and you should see something like this logged:

```
[2012-12-11 18:06:29,390]  INFO - APIDeployer API:
echoAPI has been updated from the file:
/home/soa/wso2esb-
4.5.1/repository/deployment/server/synapse-
configs/default/api/echoAPI.xml
```

61. Go to the ESB console and you should see the API listed in the APIs section.

62. Test the new service: Browse http://localhost:8281/echo/paul and you should see:



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<ns:echoStringResponse xmlns:ns="http://echo.services.core.carbon.wso2.org">
   <return>paul</return>
 </ns:echoStringResponse>
```

63. Remember that Starbucks SOAP service we installed? We are now going to look at how to map that into REST based on this article: http://wso2.org/library/articles/2012/09/get-cup-coffee-wso2-way/

64. Download the Starbucks ESB config:

65. Unzip the ESB configs into the right place:

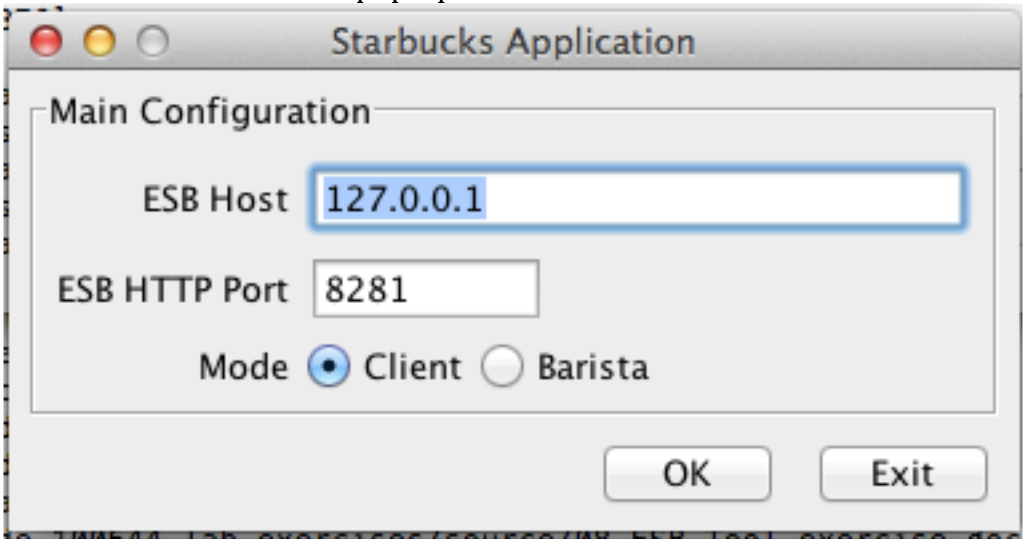
    On the command line:
    ```
    cd ~/oxsoa/wso2esb-4.5.1
    cd repository/deployment/server
    unzip ~/Downloads/starbucks-synapse-configs.zip
    ```

    When prompted:
    ```
    replace synapse-configs/default/synapse.xml? [y]es,
    [n]o, [A]ll, [N]one, [r]ename:
    ```
    Type A and hit Enter

66. Look at the ESB command line window and see how the progress is going. You should see hot deployment messages.

67. Download and start the Java App that uses the REST API:
    ```
    cd
    cd oxsoa
    wget https://svn.wso2.org/repos/wso2/people/hiranya/rest-sample/bin/starbucks-rest-sample.jar
    java –jar starbucks-rest-sample.jar
    ```

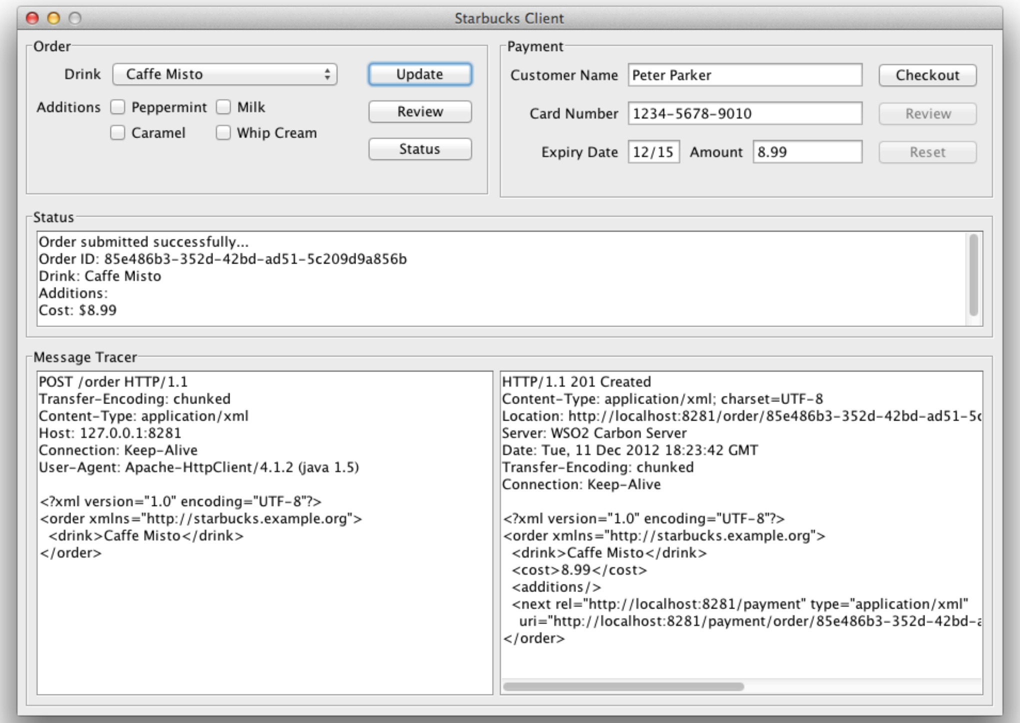
68. You should see a window pop-up:



ESB Host: 127.0.0.1
ESB HTTP Port: 8281
Mode: [✓ Client] Barista

69. Choose Client mode and click OK

70. Create some orders ☺



Drink: Caffe Misto
Additions: Peppermint, Milk, Caramel, Whip Cream
Customer Name: Peter Parker
Card Number: 1234-5678-9010
Expiry Date: 12/15 Amount: 8.99

```
Order submitted successfully...
Order ID: 85e486b3-352d-42bd-ad51-5c209d9a856b
Drink: Caffe Misto
Additions:
Cost: $8.99
```

```
POST /order HTTP/1.1
Transfer-Encoding: chunked
Content-Type: application/xml
Host: 127.0.0.1:8281
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.2 (java 1.5)

<?xml version="1.0" encoding="UTF-8"?>
<order xmlns="http://starbucks.example.org">
  <drink>Caffe Misto</drink>
</order>
```

```
HTTP/1.1 201 Created
Content-Type: application/xml; charset=UTF-8
Location: http://localhost:8281/order/85e486b3-352d-42bd-ad51-5c
Server: WSO2 Carbon Server
Date: Tue, 11 Dec 2012 18:23:42 GMT
Transfer-Encoding: chunked
Connection: Keep-Alive

<?xml version="1.0" encoding="UTF-8"?>
<order xmlns="http://starbucks.example.org">
  <drink>Caffe Misto</drink>
  <cost>8.99</cost>
  <additions/>
  <next rel="http://localhost:8281/payment" type="application/xml"
    uri="http://localhost:8281/payment/order/85e486b3-352d-42bd-a
</order>
```

71. Read the article to understand how this is working. You can also view the API configuration and definitions from the ESB Console.