**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

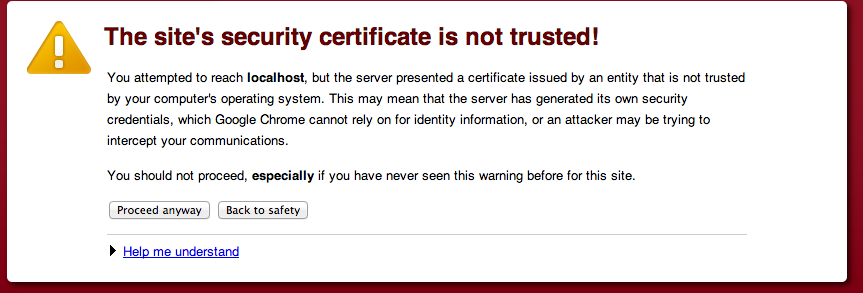
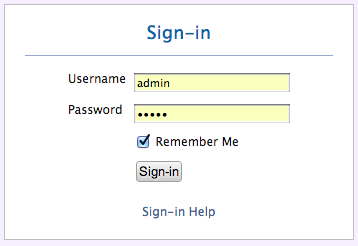
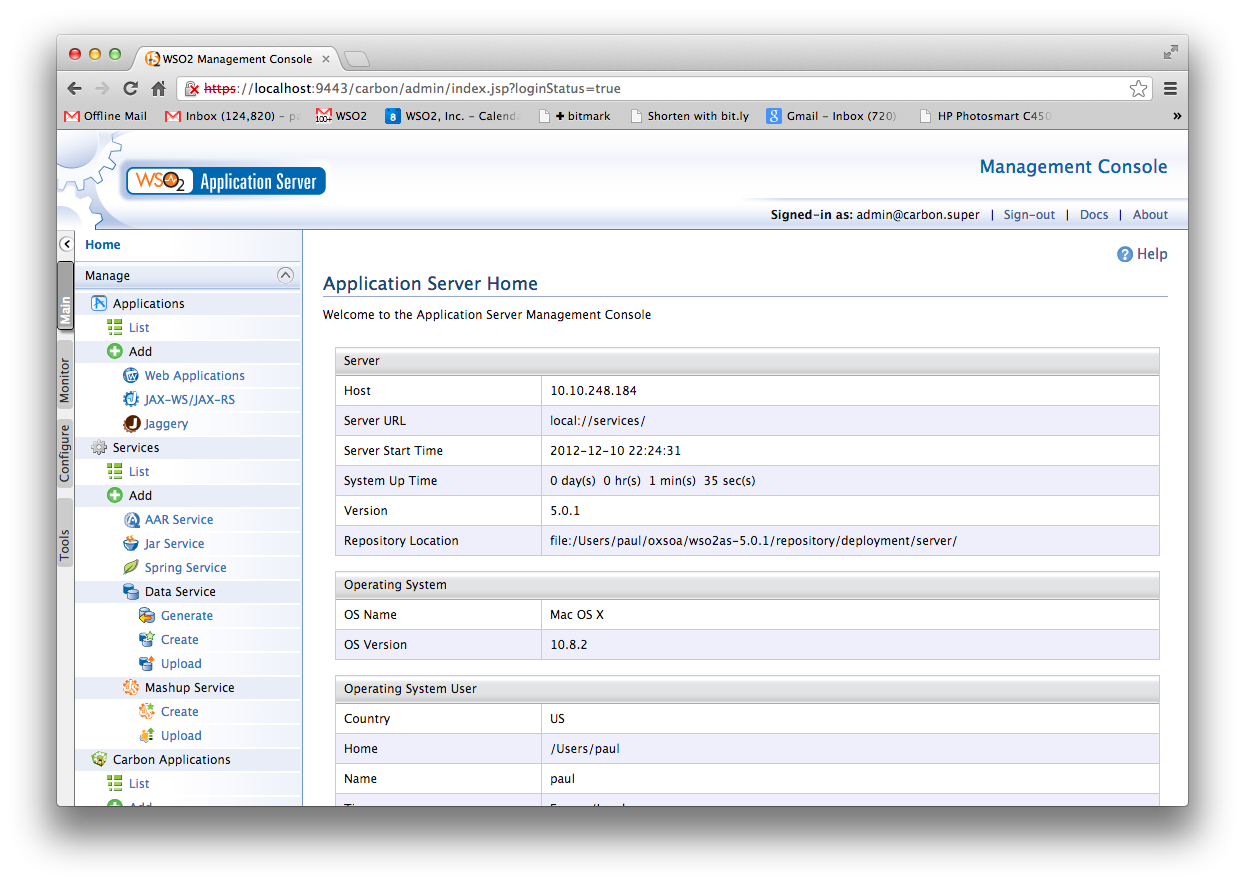
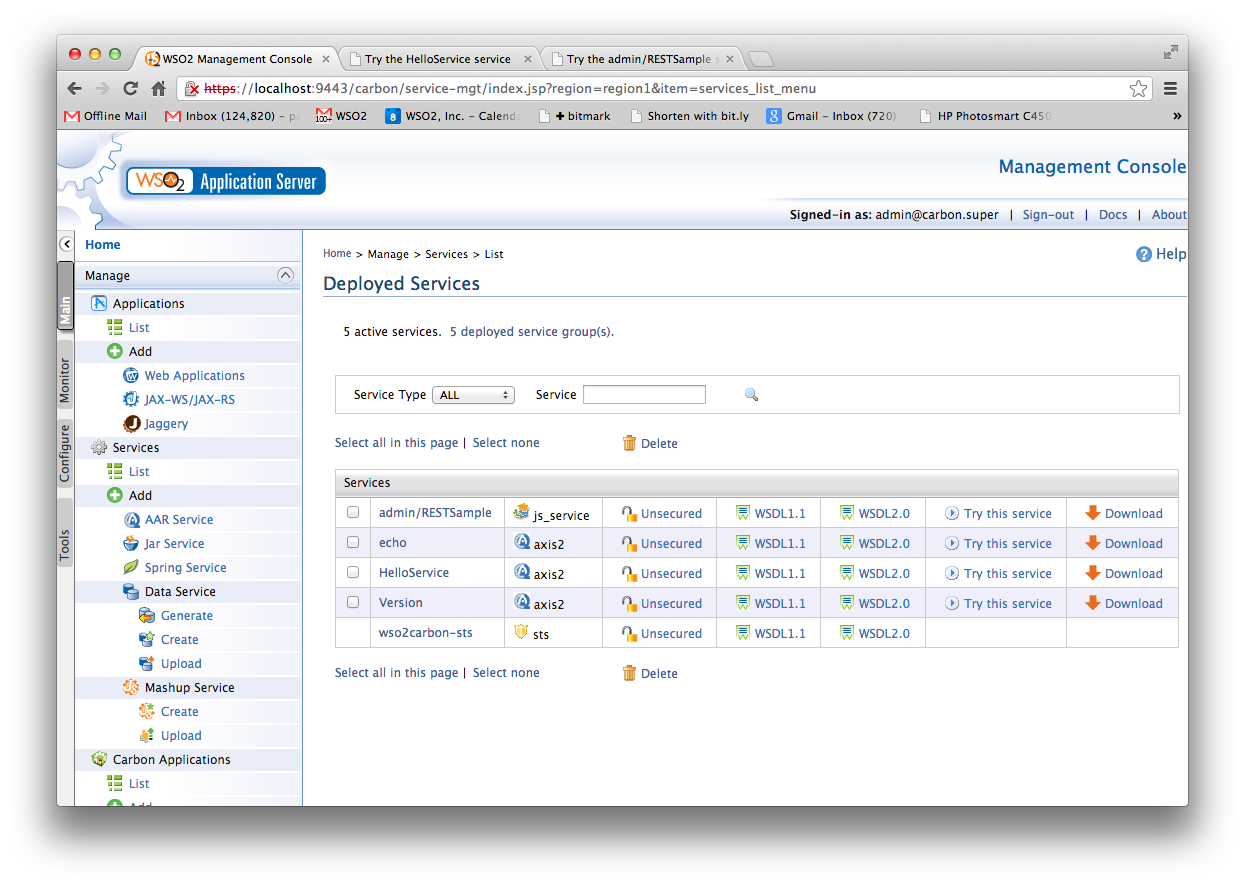
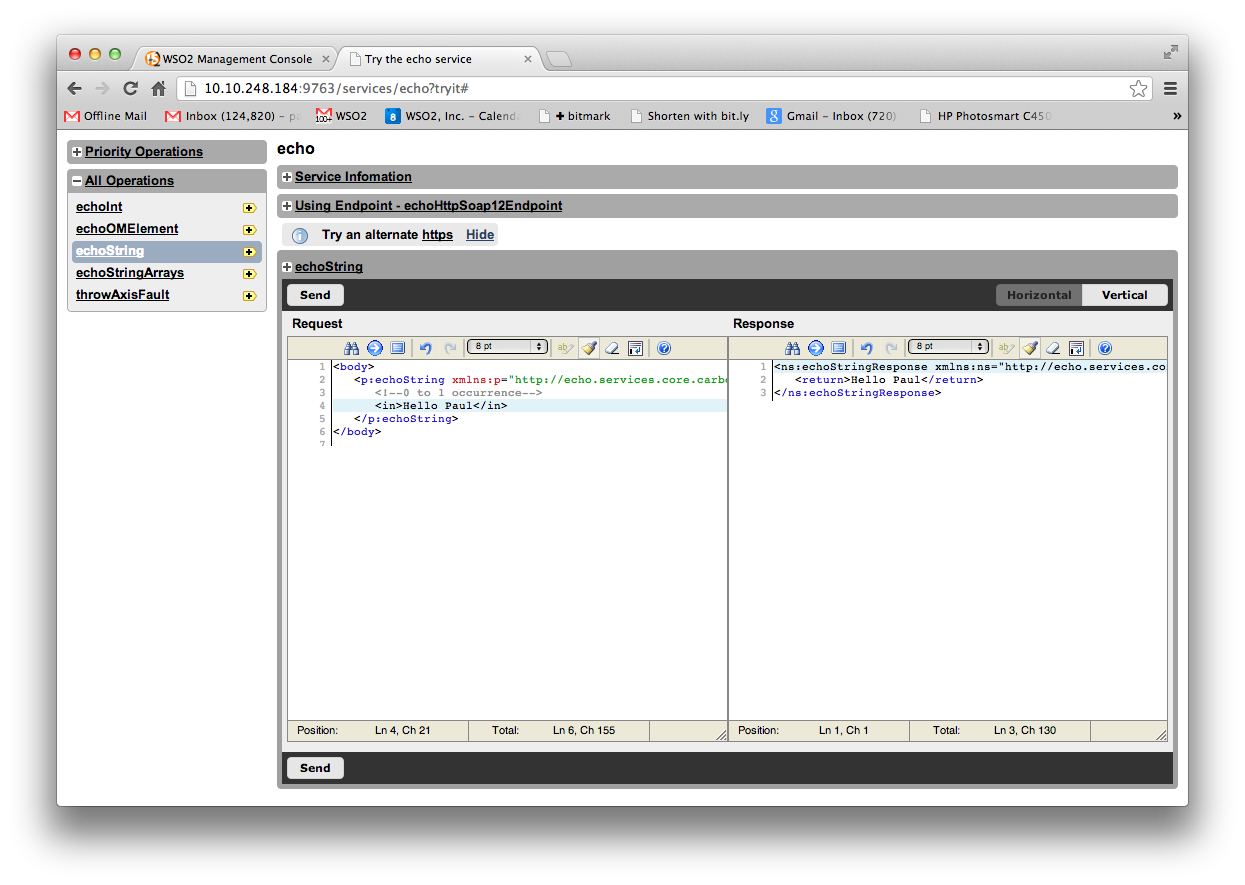
**Objectives**

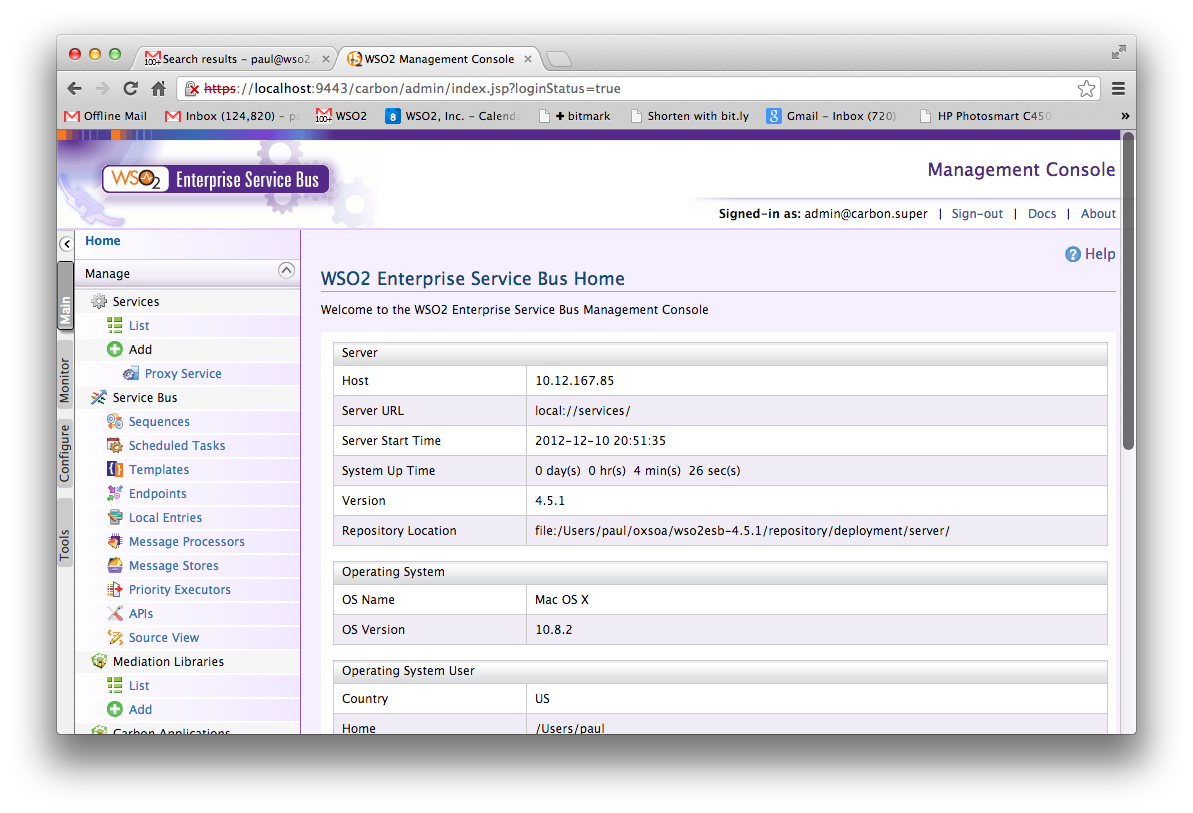
See basic ESB operations. Look at the mapping of REST to SOAP

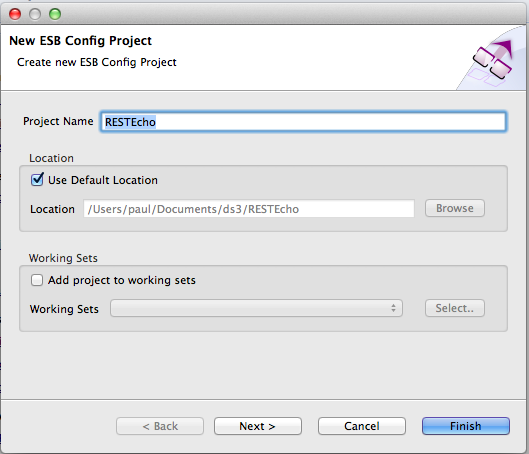
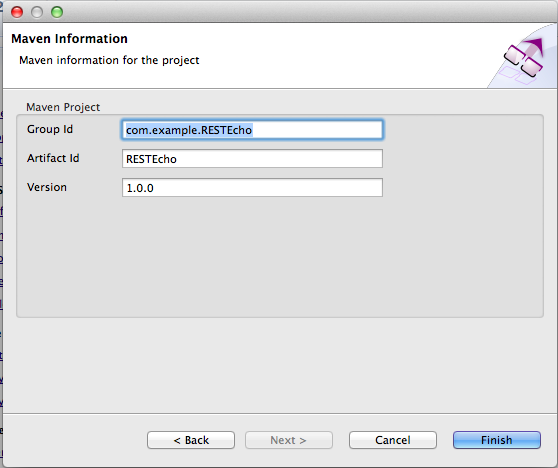
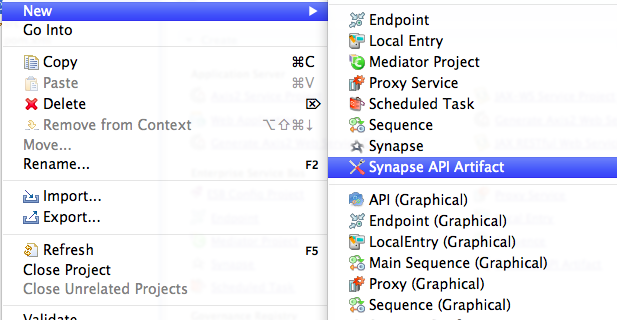
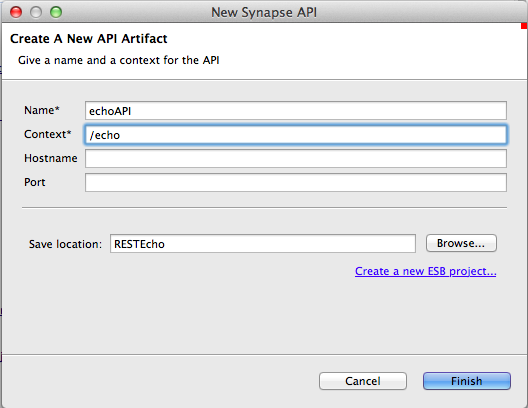
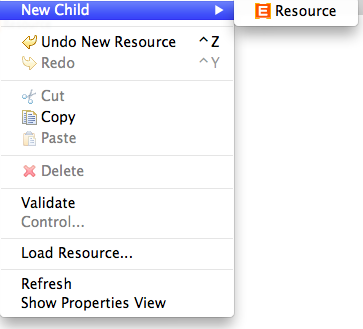
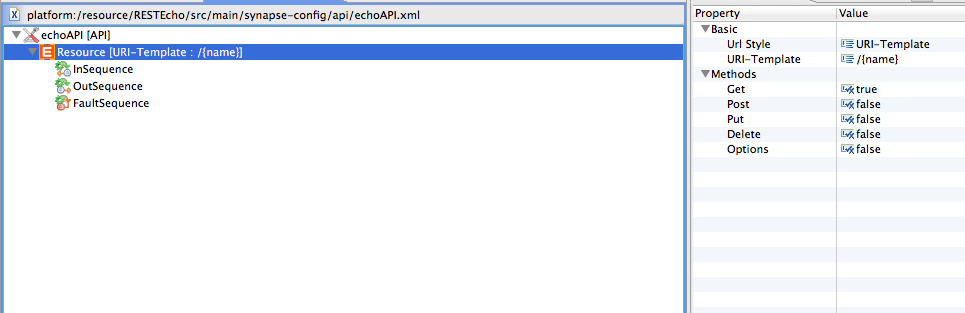
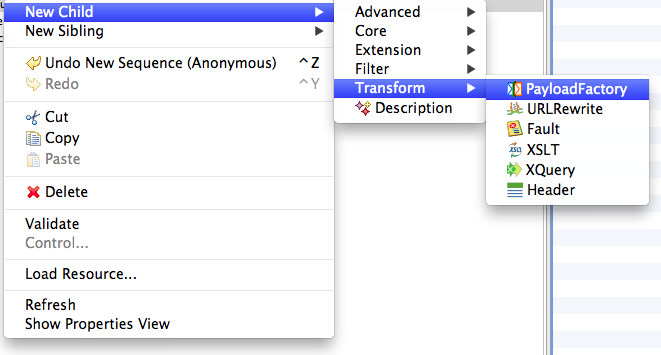
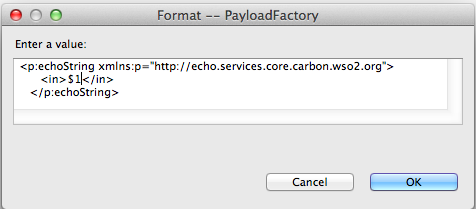
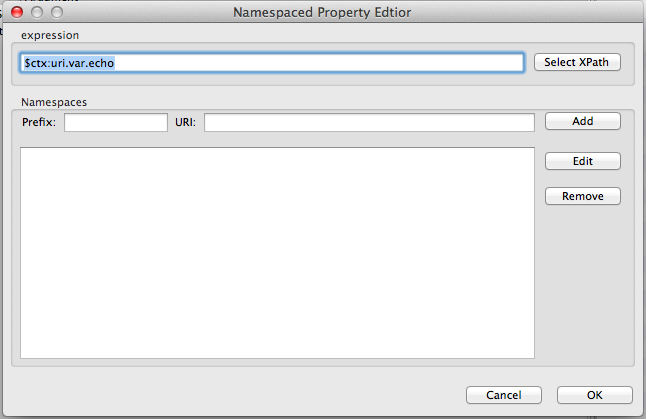
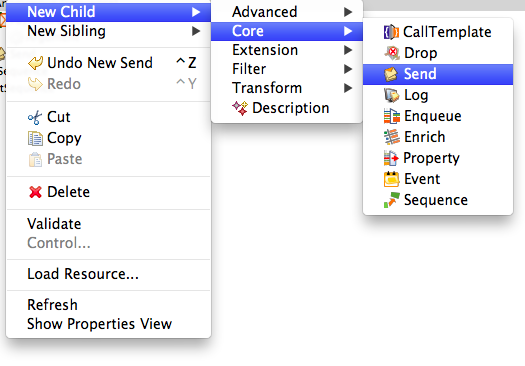
**Software Requirements**

(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   
   [http**s**://localhost:9443](https://localhost:9443) [Note the SSL URL]  
     
   You will get a message like this:   
   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:   
   
6. You should see a Web Console like this: 
7. Click on **Services->List** in the left hand menu. You should see something like this: 
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. 
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   
    [http**s**://localhost:9444](https://localhost:9444) [Note the SSL URL]

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

1. Now we are going to create a simple REST to SOAP mapping. We will initially use the echo service available in the AppServer.
2. Start or switch to Eclipse
3. Goto the Developer Studio Dashboard
4. Click **ESB Config Project**
5. Use the name **RESTEcho**
6. Click **Next**
7. Leave the Maven info the same:  
   
8. Click **Finish**
9. Right Click on the project and **New->Synapse API Artifact** ****
10. Use:  
      
    Name: **echoAPI**Context: **/echo**  
    
11. **Finish**
12. Right click on the echoAPI item.
13. Choose New Child->Resource: 
14. 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.
15. Expand the little arrow next to the Resource: 
16. Right click on InSequence and New Child-> Sequence (Anonymous)
17. Right click on the anonymous sequence and add a   
    New Child->Transform->PayloadFactory:  
    
18. Click on the PayloadFactory mediator and using the properties pane on the right hand side, edit the “Format” replacing <inline/> with the XML below.  
    
19. Add a new **Argument** child to the PayloadFactory. Use the following properties:  
    Argument Type: Expression  
    Argument Expression: $ctx:uri.var.name  
    This will grab the {name} property from the incoming URL and map it to argument $1, which will be embedded in the XML.
20. Now add a new Send mediator below the PayloadFactory. You can do this by creating a new child on the sequence.
21. On a Send mediator create a new Child -> Address Endpoint
22. Change the properties on the Address Endpoint:  
    Address: http://127.0.0.1:9763/services/echo  
    Message Format: SOAP1.1
23. 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

1. 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>

1. 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.
2. 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.
3. Instead of doing the normal code deploy

2. Download the Starbucks ESB config:
3. Unzip the ESB configs into the right place:  
     
   On the command line:  
   cd ~/oxsoa/wso2esb-4.5.1  
   cd repository/deployment/server  
   unzip ~/Downloads/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**
4. Look at the ESB command line window and see how the progress is going.