UNIVERSITY of STIRLING

Web Services

Practical 1 – Setup and Hello World

This lab contains a checkpoint towards the end. Please contact any lecturing staff when you reach this point to receive your credit.

It is important that you use the lab session to ask staff any questions you might have on the material. There are no tutorials!
Please do not sit and be stuck!

This practical will allow you to understand the installation process of a web service engine (Axis2) and how a basic web service is compiled, deployed and executed through a client invocation request. You will confirm the communication messages between client and server using Wireshark.

For the web service labs, we will be using a web service engine deployed with Tomcat. Tomcat serves as the host of the web service engine. We will use Axis2 as the web service engine. For these labs it is necessary that you have an individual Tomcat/Axis2 installation. This will allow you to have management access to the server and configure (and restart) the server and deployed services as needed.

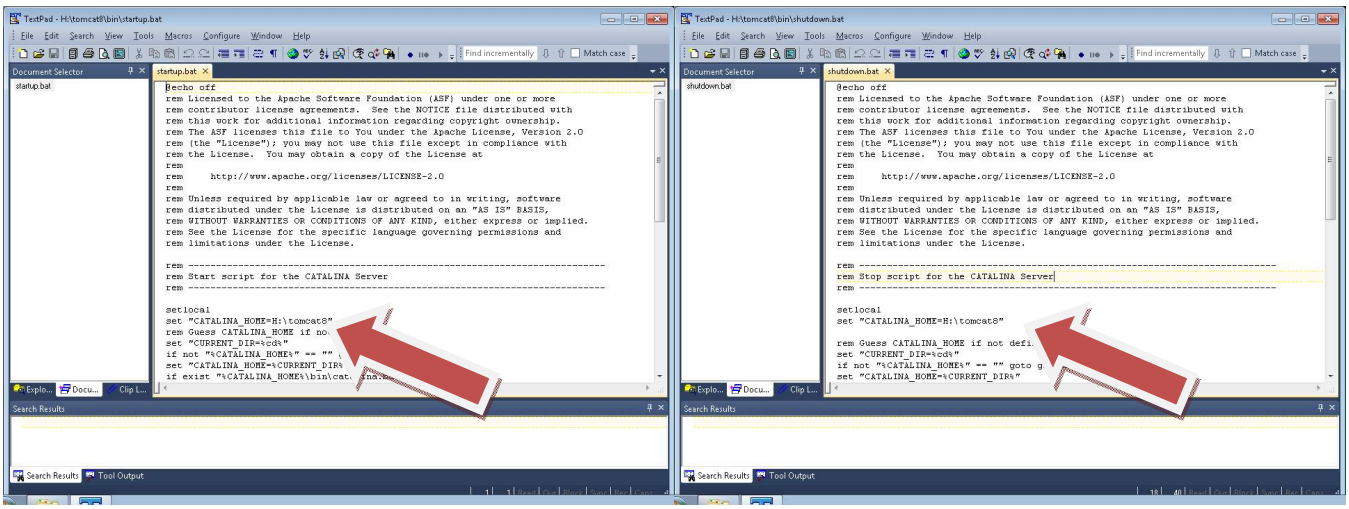
Installation and Configuration of Tomcat

A compressed Tomcat directory tree is available on the Canvas page. Copy this to your own filespace and extract the zip to a tomcat8 folder into your H: drive. You will then need to do some basic configuration by editing 'bin\startup.bat' and 'bin\shutdown.bat' to add

set "CATALINA_HOME=H:\tomcat8"

up at the top somewhere. We strongly recommend that you use Textpad for this. In any case, the use of Wordpad for this is to be avoided! There are some screenshots for this step on the next page.

Configuration of startup.bat and shutdown.bat:

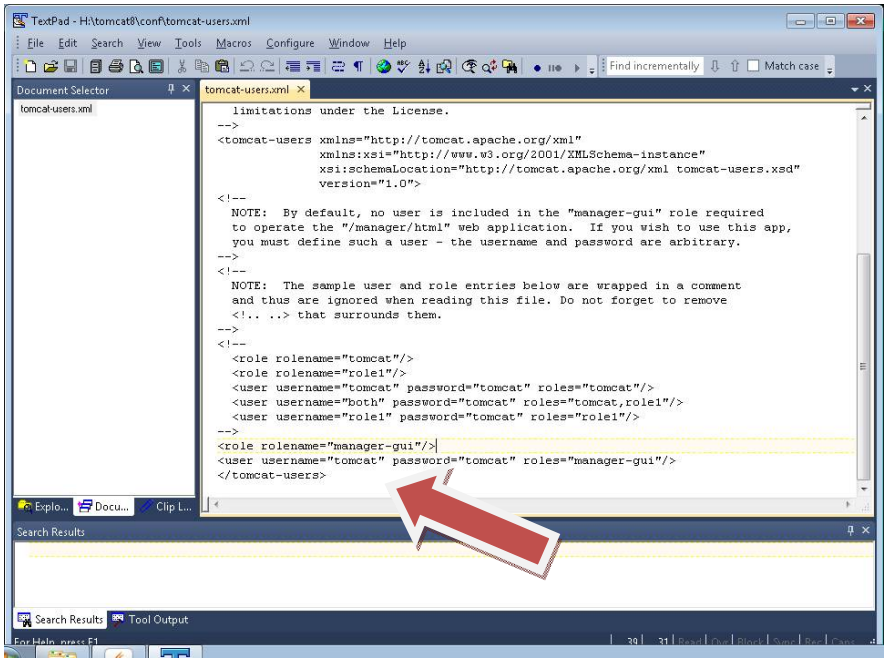


Next we need to configure a admin user with username and password. This is done in the file H:\tomcat8\conf\tomcat-users.xml. Open this file in Textpad and add the following three lines at the end of the file:

```
<role rolename="manager-gui"/>
<user username="tomcat" password="tomcat" roles="manager-gui"/>
</tomcat-users>
```

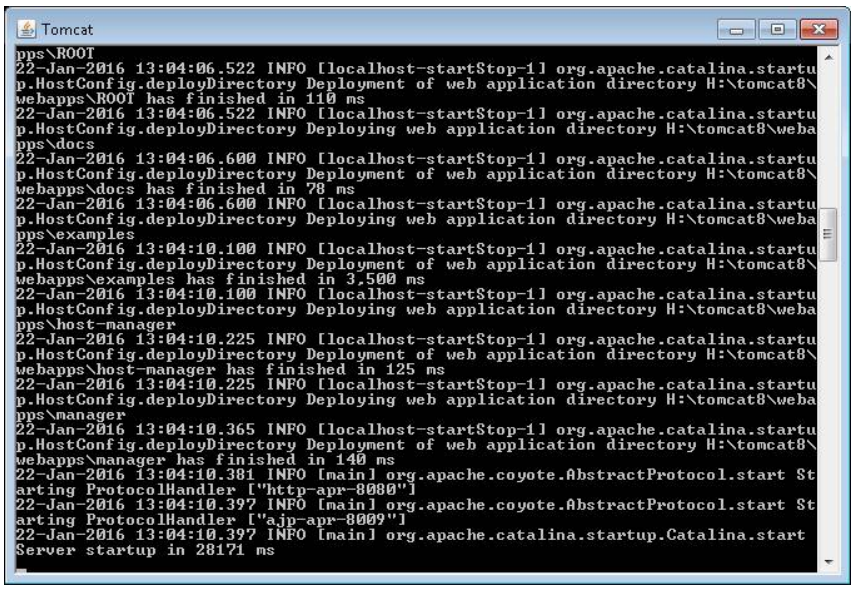
These lines configure a user *tomcat* with password *tomcat* which has access to the Tomcat Management interface. There is a screenshot of this step below.

Adding a manager user:



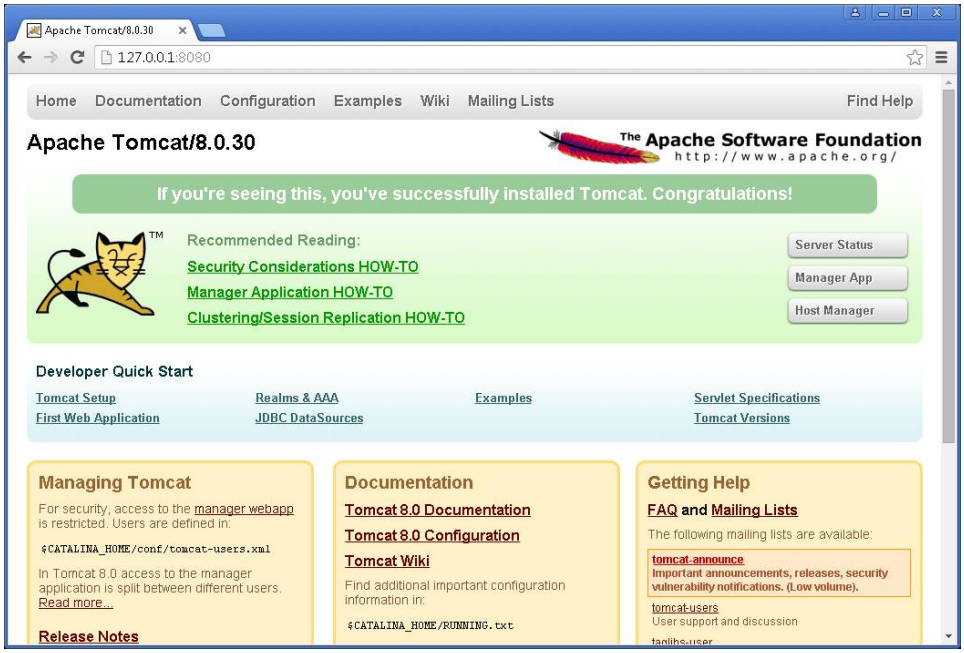
You should then be able to startup your Tomcat instance, but running startup.bat. You can do this from the command prompt, or simply double clicking this file in the Windows File Explorer. The startup of Tomcat takes about a minute.

Startup of Tomcat:



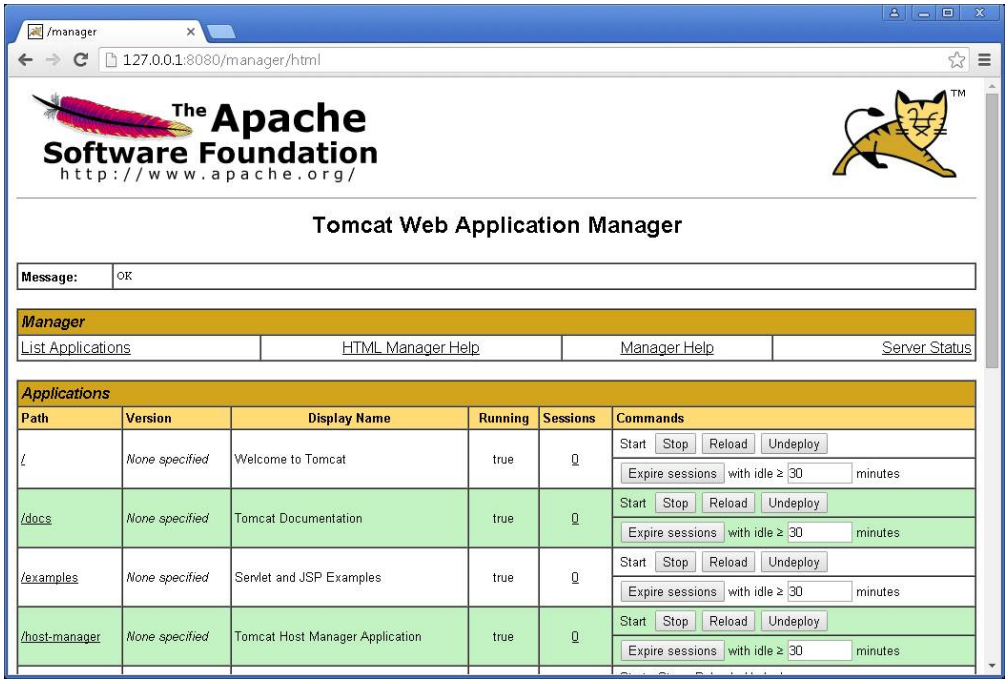
The command window which opens after you startup Tomcat usually should stay open for the duration of the lab, but you may want to minimise it.

Next check if Tomcat is running ok, and if you get access to the management interface. Open you favourite browser and point it at 127.0.0.1:8080. This refers to port 8080 on the local machine. You should see a page similar to the one below:



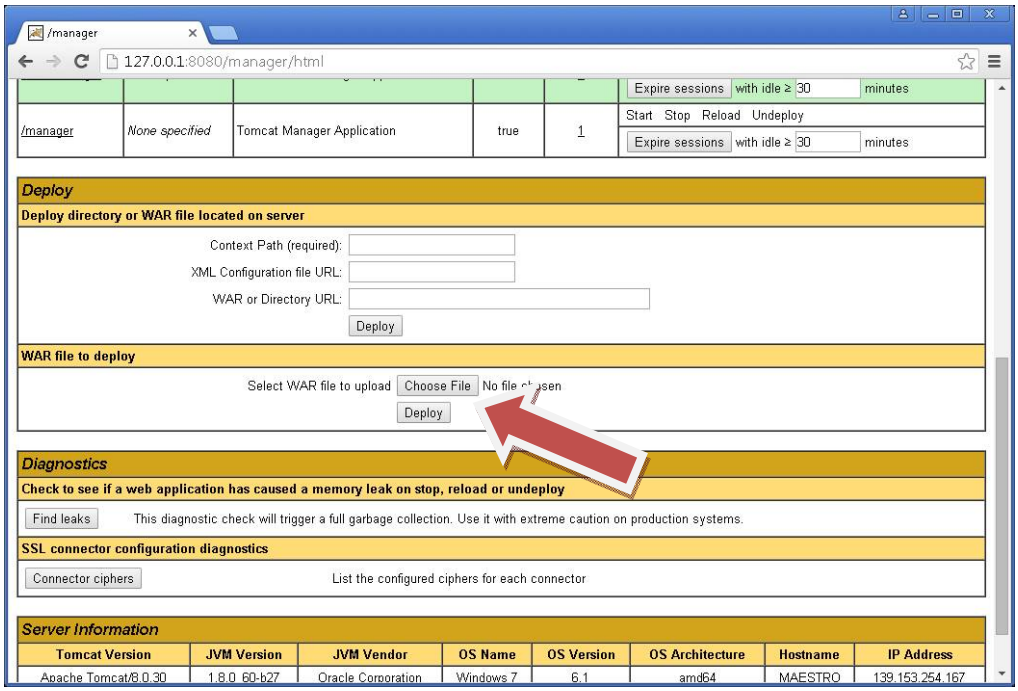
Next click on the button for “Manager App”.

Logging in with username tomcat and password tomcat should take you to a page as shown below.

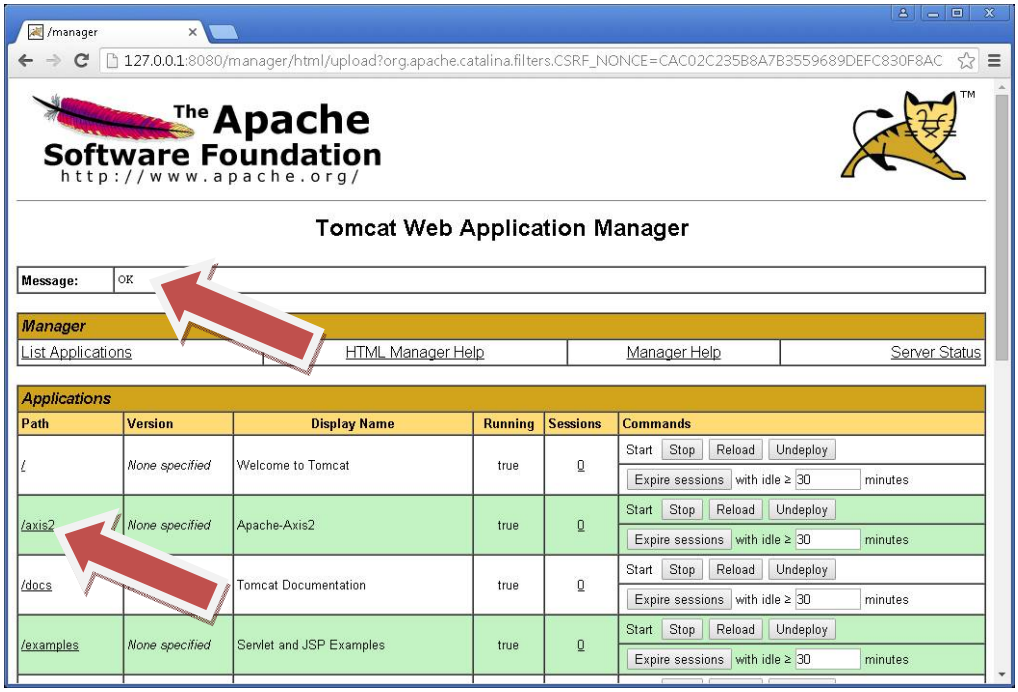


Installation and Configuration of Axis2

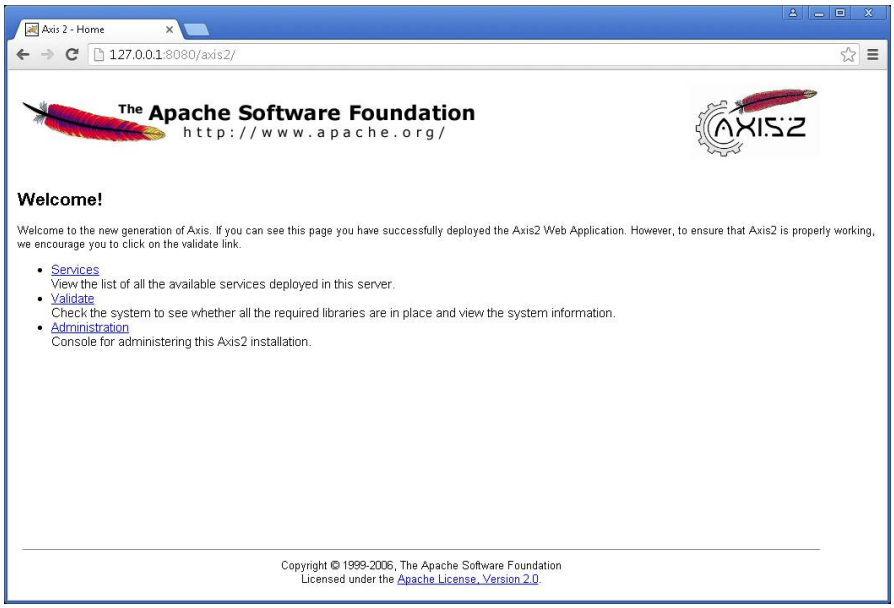
This takes you to the second step, the deployment of the Axis2 application. Scroll to about half way down this page and you should find a form to deploy a web application using a .war file.



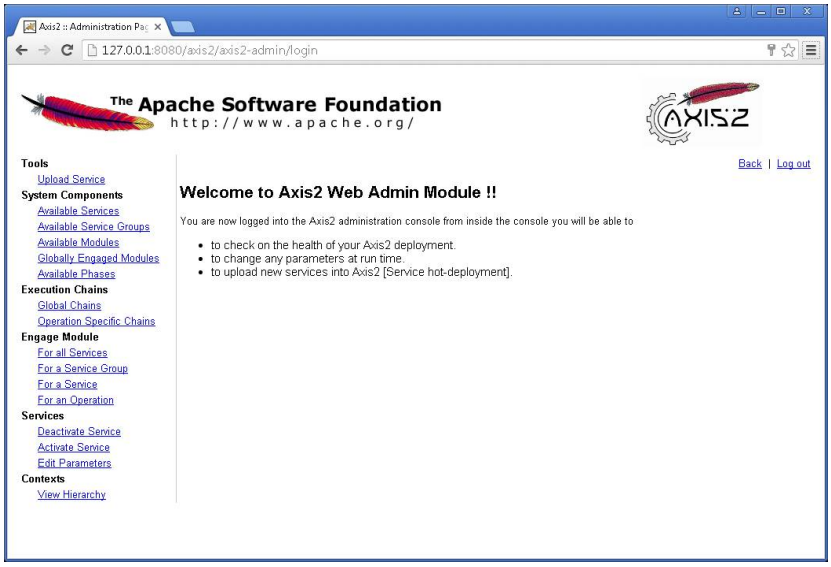
The Axis2 server is located on the Canvas page (axis2.war). Copy to your own filespace. Then select “Choose File” in the browser window and locate axis2.war. Then click Deploy to upload Axis2 to Tomcat. This step may take a minute or two. You should see a success message on completion. Axis2 should now be listed on the Tomcat management interface as an application.



Next, click on the axis2 link to open the Axis2 page in the browser (its URL for future reference is 127.0.0.1/axis2/).



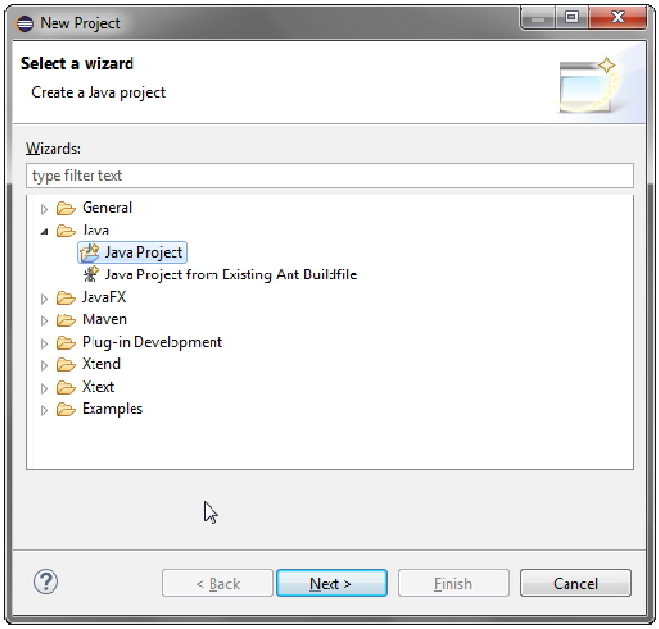
Select the Administration link and on the next page sign in as *admin* with password *axis2*.



This page confirms that Axis2 is running fine on the Tomcat server. Next we will develop a basic web service and an associated client application.

Development of an Axis2 web service

For this project we will adopt the POJO method of developing and deploying a web service. POJO stands for Plain Ordinary Java Object. So that is what this service is based on. The web service will be a basic Hello World type service. It will receive one string parameter, and convert the string to upper case characters and return the new string. We will use Eclipse as the development IDE for this project. Open up Eclipse and create a new Java project (File – New – Java Project).



Give the project a suitable name, such as HelloWorldWS.
Create a new class within this project by right-clicking on the src Folder, then choose New – Class. Keep the default package, and give the class a name, such as SimpleHelloService. Then paste the following code into the class.

```
public class SimpleHelloService {

    public String echoMsg(String arg) {
```

```
        return arg.toUpperCase();
    }
}
```

As you can see, this class has a single method echoMsg which expects a String as parameter and which returns a String value. The only operation is to convert the parameter to upper case and to return this value. Make sure your code compiles.

Next we need to define the service descriptor. Each Axis2 service must have a services.xml file which will inform Axis2 about the service. Following is the services.xml file contents for our webservice.

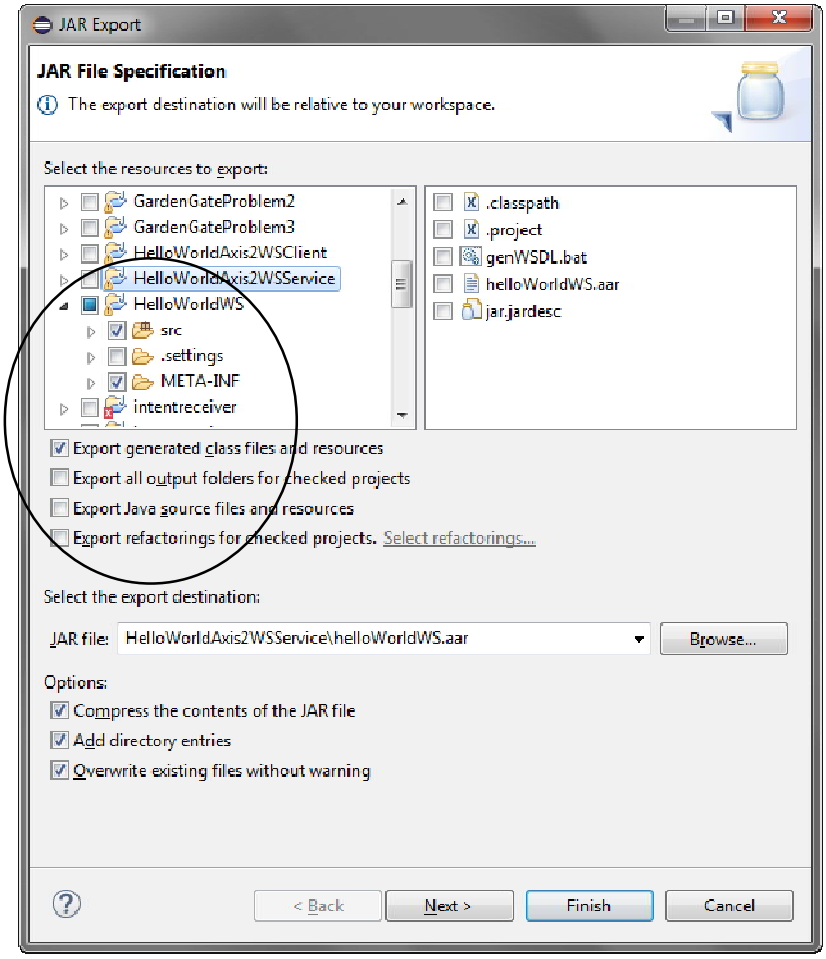
```
<service>
    <Description>This is a first Hello World Service.</Description>
    <operation name="echoMsg">
              <messageReceiver class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
      </operation>
      <parameter name="ServiceClass" locked="false">SimpleHelloService</parameter>
</service>
```

The "service" element encapsulates the information about a single service. The first child element of the "service" element is Description which simply provides a brief description of the service. The "operation" element describes the operation and the message receiver that is to be used for that operation. For this service we set the "name" attribute of the "operation" element to the name of the method that we wish to expose as a Web service operation. Axis2 provides a MessageReceiver based on Java reflection and the "messageReceiver" element declaring that org.apache.axis2.rpc.receivers.RPCMessageReceiver should be used. Finally, within the "service" element there should be a parameter specifying the service implementation Java class. The parameter is specified as a "parameter" element as shown above.

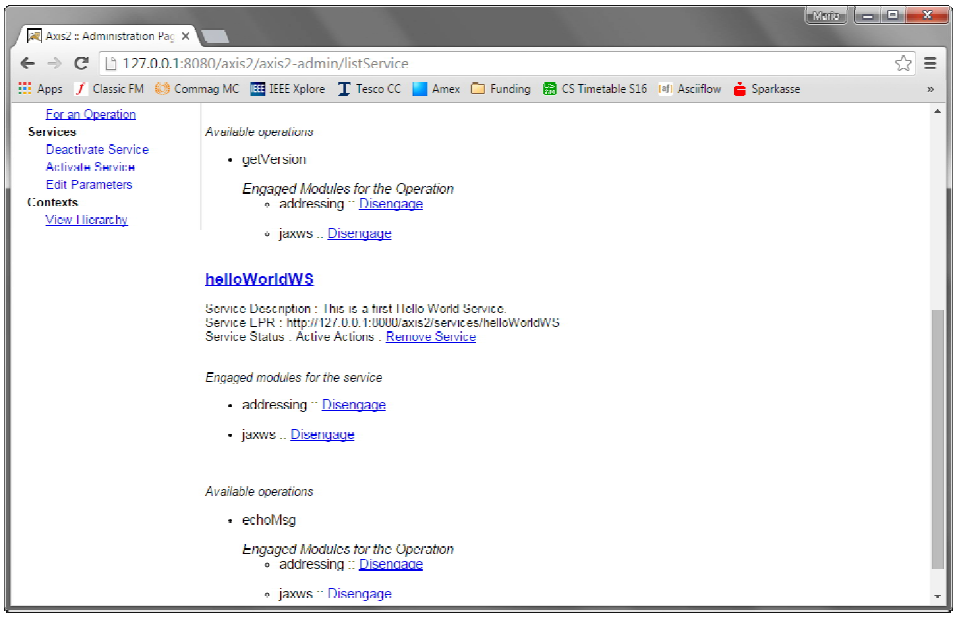
Axis2 expects services to be packaged according to a certain format. The package must be a .jar file with the compiled Java classes and a META-INF directory which will hold the services.xml file. The jar file can be name .aar to distinguish it as an Axis2 service archive. It's important to note that the part of the file name before ".aar" is the service name.

Create a Folder META-INF within your project (File – New – Folder). Within this folder create a new file services.xml (File – New – File). Paste the content from above into this xml file and save.

Next we need to package the web service as an .aar archive. This is a .jar file under a different name. Eclipse supports the creation of jars. Select File – Export – Jar File. Make sure that the correct project is selected. The src Folder and the META-INF folder should be part of the jar, and the compiled classes. See the screenshot on the next page. Give the file an appropriate name – remember, the name of the file is the name of the web service. The example below uses the name helloWorldWS.aar.

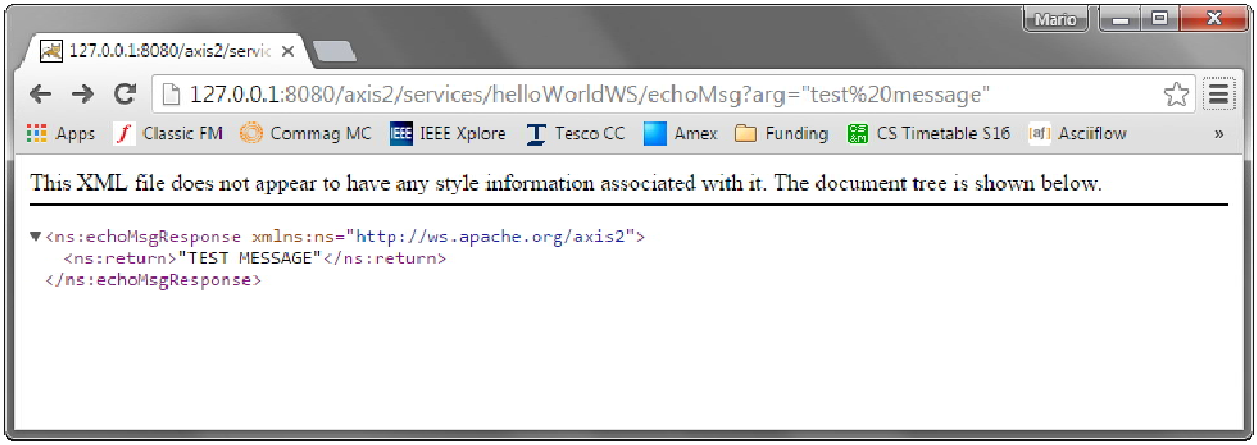


This should create the deployable web service archive. To deploy it, go back to the browser and select Upload Service, and then Choose File. Navigate to your aar file and Upload. Afterwards select Available Services and confirm your service is listed.



You have deployed your first web service! Let's test it quickly in a browser.
In a browser window, you should be able to call your service with a URL such as:

http://127.0.0.1:8080/axis2/services/helloWorldWS/echoMsg?arg=%22test%20message%22



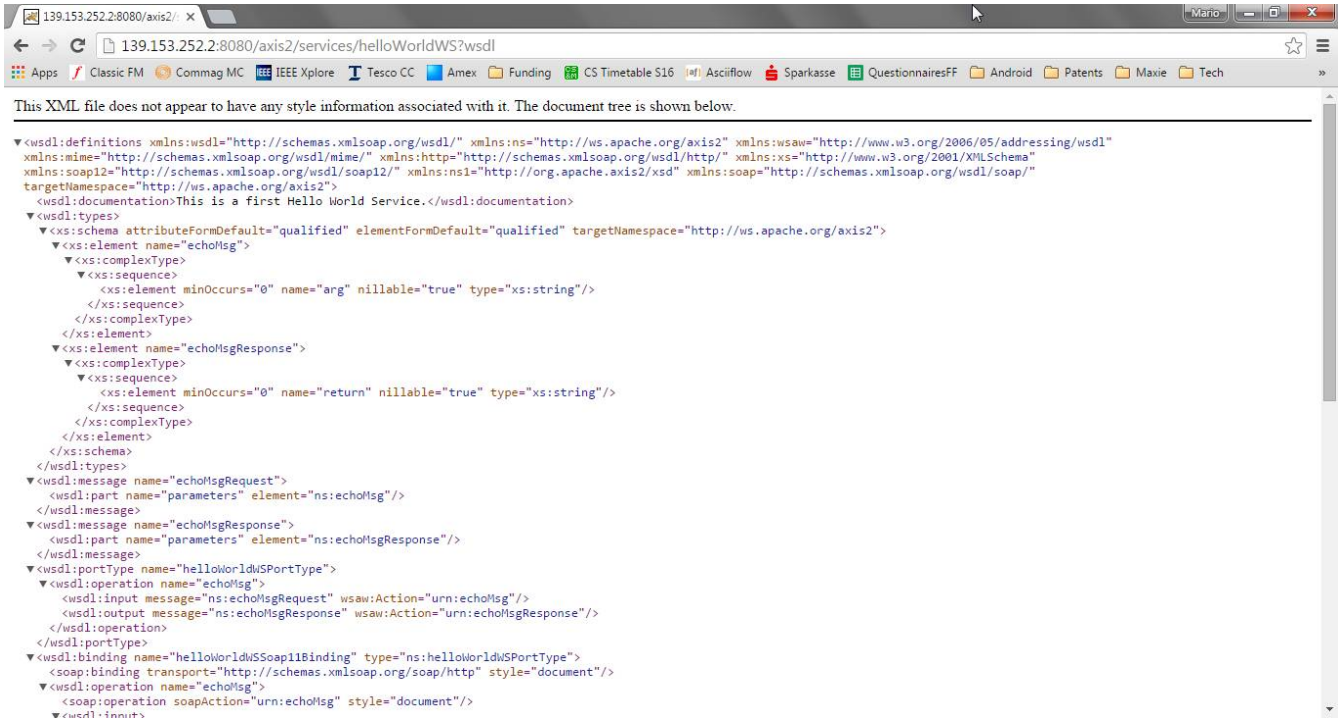
Point your browser at somebody else's server. The next step is to create a client to call this service "properly", in a programmatic way using a Java client application.

Development of an Axis2 client

Go back to eclipse and create a new project, perhaps named HelloWorldWSClient. The first step to create a client is to generate the client stubs. These are basically library entry points to generate the messages to communicate with the service. In this example the messaging will employ SOAP (Simple Object Access Protocol). There are a set of libraries and commands in an axis2 folder on K:

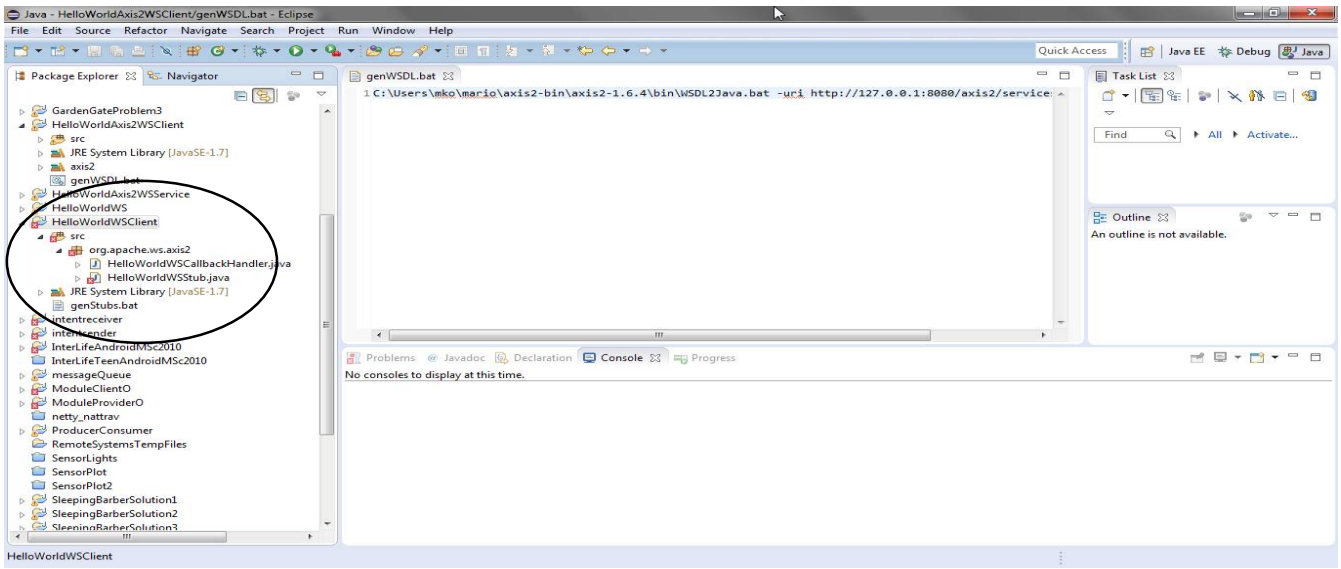
K:\CSCU9YW\axis2-1.6.4\bin\WSDL2Java.bat –uri
http://127.0.0.1:8080/axis2/services/helloWorldWS?wsdl

This command should be executed from the directory for the client created by eclipse when you created the client project. The best way to do so is to create a new file in eclipse (File – New – File) and name it something .bat. The content of this file should be the command line above (this assumes your web service is called helloWorldWS). This commands turns the WSDL (Web service description language) description of your webservice into Java code. The WSDL, you can check by clicking on the web service name in the Axis2 list of web service page (UEL = http://127.0.0.1:8080/axis2/services/helloWorldWS?wsdl). You should get something like this:

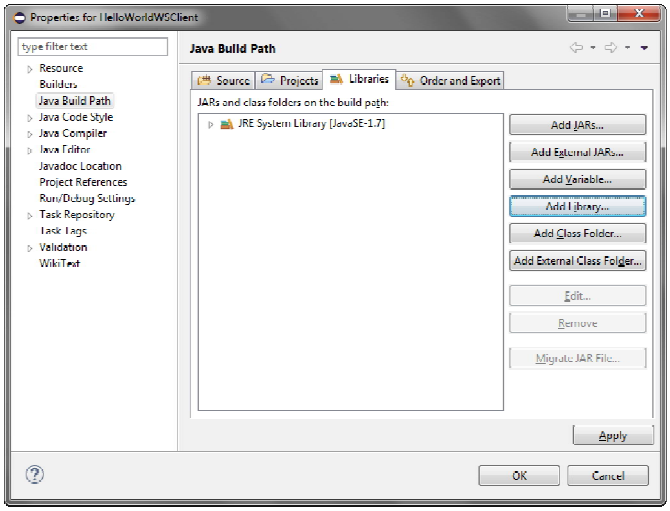


We will look more closely at WSDL in the lectures.

After you executed the command, it will have created directories and client stub files as shown below.



Note that there are compile errors. This is because the project is missing the Axis2 libraries. We will fix this in the next step. We need to configure the Build Path (right click on the project title, then Build Path, Configure Build Path).

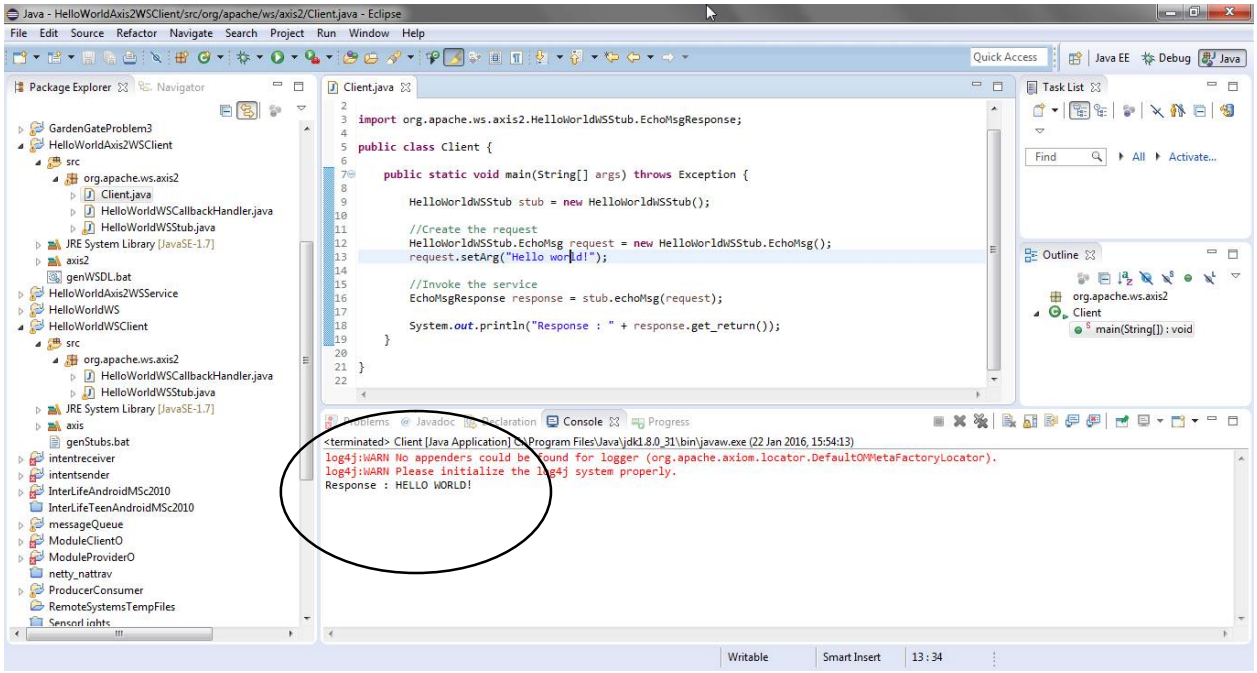


Next click Add Library and select User Library, then User Libraries and then select New. Give a name for the new library, such as axis2. Then click on Add External Jars and navigate to the K: drive and the folder axis2-bin\axis2-1.6.4\lib within the CSCU9YW directory. This contains quite a number of jars! Select them all by holding Shift and then the Arrow Down key. And Click Open and then OK in the parent window. Finally click Finish and you should be back in the Build Path window. Here you should see the new library added next to the JRE System Libraries. Once you click OK to close the Build Path window, your project should recompile and the errors gone.

We now need to create a simple Java class which calls the stubs and issues the call to the webservice. Create a new Java class Client.java in the same package as the stubs (File – New – Java Class). The code for the client is provided below. Paste this into your new class.

```
package org.apache.ws.axis2;

import org.apache.ws.axis2.HelloWorldWSStub.EchoMsgResponse;

public class Client {

    public static void main(String[] args) throws Exception {

       HelloWorldWSStub stub = new HelloWorldWSStub();
       HelloWorldWSStub.EchoMsg request = new HelloWorldWSStub.EchoMsg();
        request.setArg("Hello world!");

        EchoMsgResponse response = stub.echoMsg(request);
        System.out.println("Response : " + response.get_return());
    }
}
```

Note the HelloworldWS name and the echoMsg method name. Compile the project and run. You should get output similar to:

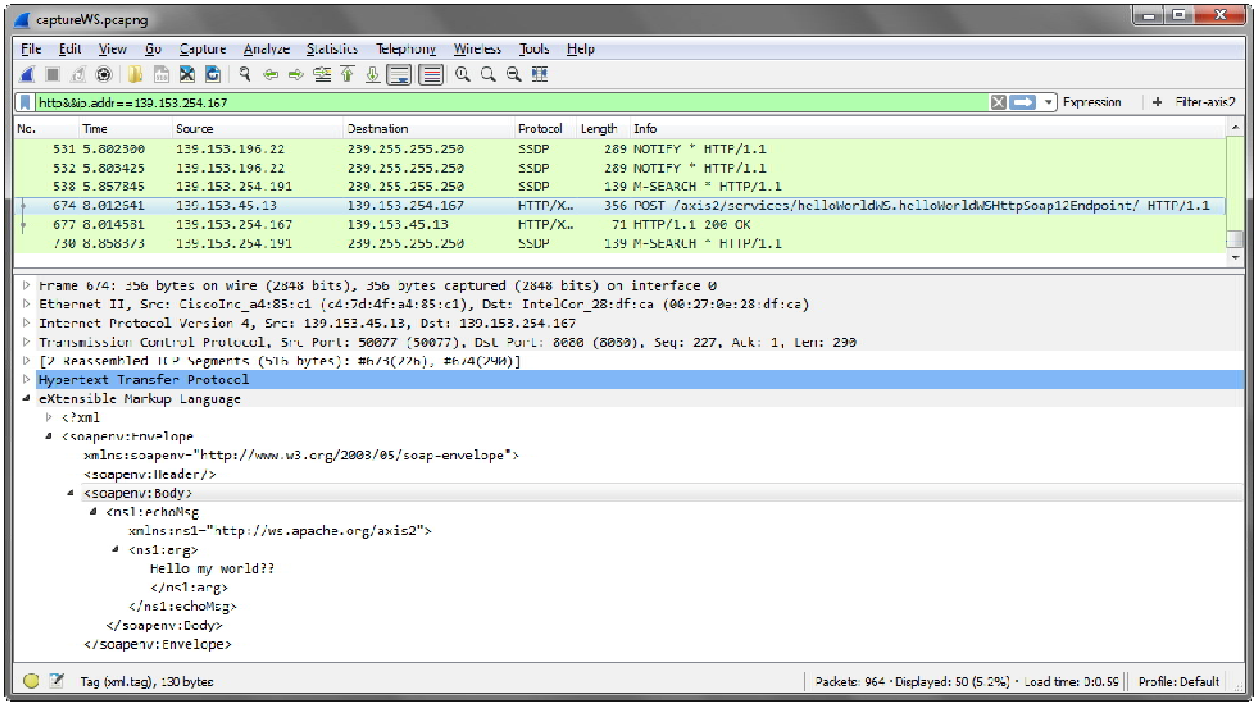


If you get output of HELLO WORLD in upper case, you have succeeded calling the webservice. Try changing the string in your client and rerun. Does the output change too?

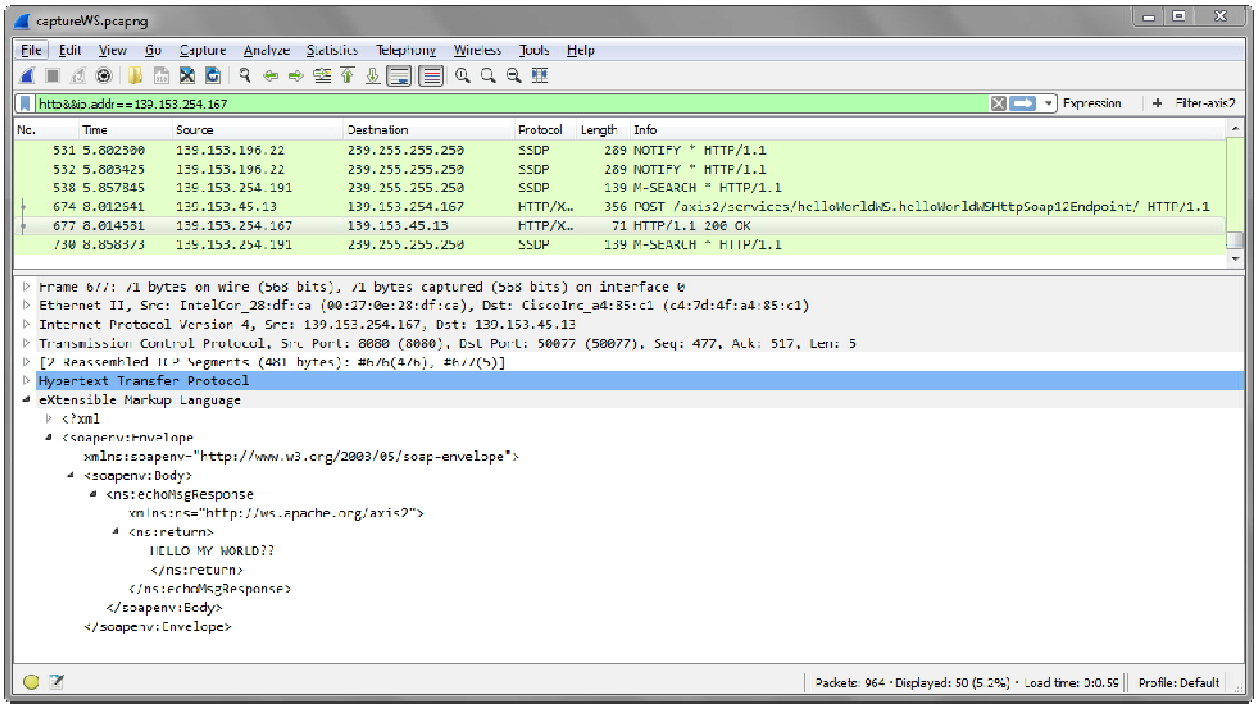
Now try your client with somebody else's server. You will need to regenerate the client stubs as the IP address will be different. For the regeneration of the stubs to be successful you may have to delete (or rename) the existing stub files (not your client.java!) as the tool may not overwrite the existing stubs. When you try the other service, make sure it manipulation of the string you pass is different to what your service is doing. Ideally, the server attaches its own name to the response message. This way you are sure that you actually called the other service.

Capturing SOAP messages between client and service

In this step, you will investigate the communication messages between client and service. For this step we are using Wireshark which you should remember from the W6 labs. Start up Wireshark and configure a filter to capture http packets arriving at your service. When get somebody to call your service. You should get a capture similar to the following screenshot.



This screenshot shows the SOAP request message. The screenshot below shows the SOAP response.



We will discuss SOAP in some detail in the lectures.

Checkpoint