**Hello world with Apache Axis2**

This is a step by step guide to hosting a simple Web service with Axis2 and interacting with that service using a client based on generated code.

We will be starting with a simple Java class which will be turned into a service with the necessary packaging. Then a client will be created using the code generated by the WSDL2Java code which will be used to invoke the service we created.

**Step 1: A simple Java class**

First we need a Web Service and an operation in that service for a client to invoke. Lets develop a Web service with an operation which will echo a string value. The simplest way to do this is with a Java class as shown below:

----------------------------------------------------------------------------------------------------------------------

/\*\*

 \* The service implementation class

 \*/

public class SimpleService {

 /\*\*

 \* The echo method which will be exposed as the

 \* echo operation of the web service

 \*/

 public String echo(String value) {

 return value;

 }

}

----------------------------------------------------------------------------------------------------------------------

save as - SimpleService.java

----------------------------------------------------------------------------------------------------------------------

**Step 2: 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 the SimpleService web service.

<service>

 <parameter name="ServiceClass" locked="false">SimpleService</parameter>

 <operation name="echo">

 <messageReceiver class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>

 </operation>

</service>

the "service" element encapsulates the information about a single service. 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 below.

<parameter name="ServiceClass" locked="false">SimpleService</parameter>

The second child element of the "service" element "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. Hence we set it to "echo":

<operation name="echo">

Axis2 provides a MessageReceiver based on Java reflection and the "messageReceiver" element declaring that org.apache.axis2.rpc.receivers.RPCMessageReceiver should be used.

<messageReceiver class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>

**Step 3: Packaging the service**

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. Its important to note that the part of the file name before ".aar" is the service name.

Create a temp directory in the same location where the SimpleService.java file exists

[Linux]

mkdir temp

[Windows]

md temp

Now compile the SimpleService.java class and move the SimpleService.class file to the temp directory.

**javac SimpleService.java -d temp/**

Create a META-INF directory within the "temp" directory and copy the service.xml file into the META-INF directory.

Change directory to the "temp" directory and use the "jar" command as follows to create the service archive named SimpleService.aar.

**jar cvf SimpleService.aar .**

**Step 4: Hosting the service**

There are a number of ways to host the service that was created in the previous step. The two main methods are

* Using the SimpleHTTPServer that is available in the Axis2 distribution
* Using Axis2 with Tomcat

This tutorial will use the org.apache.axis2.transport.http.SimpleHTTPServer to host the SimpleService.aar

Axis2 service archives are placed in a directory named "services" in a repository directory. The structure of an example repository directory is shown below.



Now create the my-axis2-repo directory structure and copy the SimpleService.aar file into the "services" directory. This example does not require the axis2.xml to be available in the "conf" directory.

Now we have to start the SimpleHTTPserver using the above my-axis2-repo directory as the repository directory. The axis2-std-1.0-RC1-bin distribution comes with a "bin" directory which contains a Linux shell script and a Windows batch file to start the SimpleHTTPServer: http-server.sh and http-server.bat

Start the server pointing to my-axis2-repo directory:

[Linux]

**sh http-server.sh /path/to/my-axis2-repo**

[Windows]

**http-server.bat drive:\path\to\my-axis2-repo**

The following output will be shown in the console:

[SimpleHTTPServer] Starting

[SimpleHTTPServer] Using the Axis2 Repository /home/ruchith/Desktop/ibm-workshop/axis2-repo

[SimpleHTTPServer] Listening on port 8080

[JAM] Warning: You are running under a pre-1.5 JDK. JSR175-style source annotations will not be available

[SimpleHTTPServer] Started

Now when we point a browser to http://localhost:8080/ the SimpleHTTPServer will respond with a list of available services and the SimpleService will be listed there.



**Step 5: Accessing the service with a generated client**

Now lets use the WSDL2Java tool generate the client side stubs to interact with the service.

[Linux]

**$ sh WSDL2Java.sh -uri http://localhost:8080/axis2/services/SimpleService?wsdl -o /path/to/my/client/code/**

[Windows]

**WSDL2Java.bat -uri http://localhost:8080/axis2/services/SimpleService?wsdl -o /path/to/my/client/code/**

This generates two .java files and we will be using the org.apache.axis2.SimpleServiceStub to invoke the "echo" operation of the service.

Now lets create a new Client.java class which uses the org.apache.axis2.SimpleServiceStub. For simplicity lets create the Client.java file in the same package as the generated code (i.e. org.apache.axis) and save it along with the other generated code.

----------------------------------------------------------------------------------------------------------------------

package org.apache.axis2;

import org.apache.axis2.SimpleServiceStub.EchoRequest;

import org.apache.axis2.SimpleServiceStub.EchoResponse;

public class Client {

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

 SimpleServiceStub stub = new SimpleServiceStub();

 //Create the request

 EchoRequest request = new SimpleServiceStub.EchoRequest();

 request.setParam0("Hello world");

 //Invoke the service

 EchoResponse response = stub.echo(request);

 System.out.println(response.get\_return());

 }

}

----------------------------------------------------------------------------------------------------------------------

save as - Client.java

----------------------------------------------------------------------------------------------------------------------

Now to we can compile and run the client code. Its is important to note that the classpath must have all the jars in the "lib" directory of the axis2-std-1.0-RC1-bin distribution when compiling and running the client.

The following command will compile the client's source to a "temp" directory

**$ javac -extdirs /path/to/axis2-RC1-std-bin/lib/ org/apache/axis2/\*.java -d temp/**

We can run the client program as shown below from the "temp" directory:

**$ java -Djava.ext.dirs=/path/to/axis2-RC1-std-bin/lib/ org.apache.axis2.Client**

The output will be :

Response : Hello world

**Step 6: Monitoring the messages**

To view the request and response SOAP messages we can use the tcpmon tool.

We can start the SimpleHTTPServer on port 9090 and make the tcpmon listen on port 8080 and forward the requests to port 9090.

Using "-p9090" as an additional argument in starting the SimpleHTTPServer we can start it on port 9090.

Example:

**sh http-server.sh /path/to/my-axis2-repo -p9090**

Now when we run the client once again we can view the messages.

