Consuming webservices in JSP with Apache AXIS

Last modified: 10/19/2004

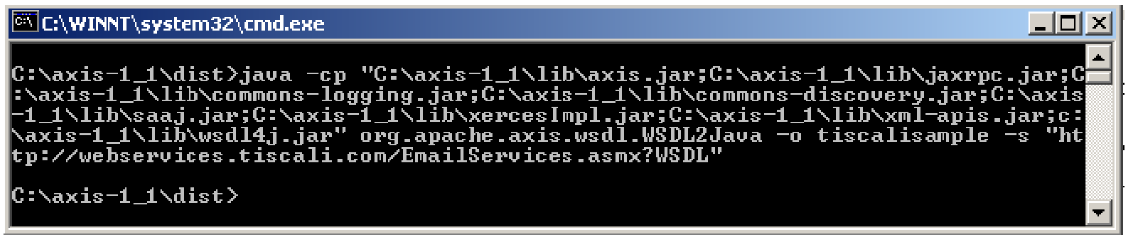
Recently I started fumbling around with webservices, a nice integration and communication tool for my customers, what follows is a work in progress, which is going to be enhanced while I proceed in my “trial and error for the illiterate” learning path.

First of all I installed AXIS (http://ws.apache.org/axis/) from the Apache Software foundation and, after validating the installation with the “happyAxis” page, had a working environment for my tests.

The first thing that came to my mind was consuming an external webservice, and I chose to experiment with Tiscali’s public webservices (http://webservices.tiscali.com/).

Axis is able to generate a client for you and I decided to go for it!

It worked automagically, just typed (on my customer’s PC AXIS is located in c:\axis_1-1\):



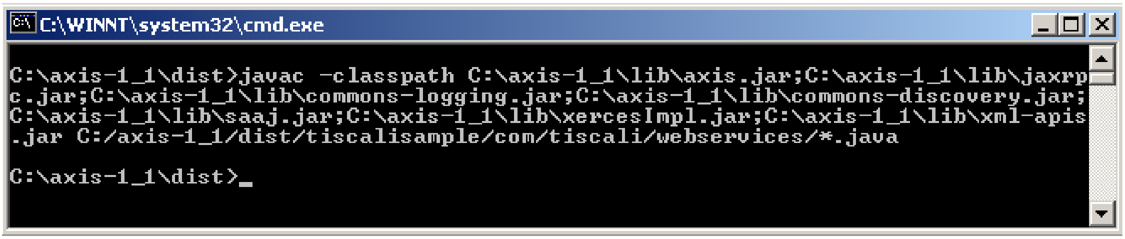
```
C:\axis-1_1\dist>java -cp "C:\axis-1_1\lib\axis.jar;C:\axis-1_1\lib\jaxrpc.jar;C
:\axis-1_1\lib\commons-logging.jar;C:\axis-1_1\lib\commons-discovery.jar;C:\axis
-1_1\lib\saaj.jar;C:\axis-1_1\lib\xercesImpl.jar;C:\axis-1_1\lib\xml-apis.jar;c:
\axis-1_1\lib\wsdl4j.jar" org.apache.axis.wsdl.WSDL2Java -o tiscalisample -s "ht
tp://webservices.tiscali.com/EmailServices.asmx?WSDL"

C:\axis-1_1\dist>
```

And in folder ./tiscalisamples (specified with the –o option) some folders and java files were generated:

./tiscalisample/**com/tiscali/webservices/**some java files :

1. ArrayOfServiceTest.java
2. deploy.wsdd
3. ServiceTest.java
4. SmtpResult.java
5. Tiscali_x0020_Email_x0020_Services.java
6. Tiscali_x0020_Email_x0020_ServicesLocator.java
7. Tiscali_x0020_Email_x0020_ServicesSoap.java
8. Tiscali_x0020_Email_x0020_ServicesSoapImpl.java
9. Tiscali_x0020_Email_x0020_ServicesSoapStub.java
10. undeploy.wsdd

Now I compiled the whole lot :



```
C:\axis-1_1\dist>javac -classpath C:\axis-1_1\lib\axis.jar;C:\axis-1_1\lib\jaxrp
c.jar;C:\axis-1_1\lib\commons-logging.jar;C:\axis-1_1\lib\commons-discovery.jar;
C:\axis-1_1\lib\saaj.jar;C:\axis-1_1\lib\xercesImpl.jar;C:\axis-1_1\lib\xml-apis
.jar C:/axis-1_1/dist/tiscalisample/com/tiscali/webservices/*.java

C:\axis-1_1\dist>_
```

And so it all ended up in a bunch of .class files in the same folder.

Then I copied the whole ./com/tiscali/webservices/*.class to the ./WEB-INF/classes folder of the standard “examples” webapp shipped with Tomcat.

Now the “real” client, those were only helper classes, I chose to write it in jsp for simplicity, it all boiled down to a few lines:

```
<%@ page contentType="text/html; charset=iso-8859-1" language="java" errorPage="" %>
<html>
<head>
<title>eMail validation consumer</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<body>
<%
com.tiscali.webservices.Tiscali_x0020_Email_x0020_ServicesLocator aLocator = new com.tiscali.webservices.Tiscali_x0020_Email_x0020_ServicesLocator();
com.tiscali.webservices.Tiscali_x0020_Email_x0020_ServicesSoap aServicesSoap = aLocator.getTiscali_x0020_Email_x0020_ServicesSoap();
String validatorResult = aServicesSoap.isValidEmail("mail@xyz.com");
out.write(validatorResult);
%>
</body>
</html>
```

Which returned “true” as this is a valid (even if probably not existent) email address.

This made me happy and prompted for further investigation of those “webservices”.

The nice thing of this code is that WSDL2Java (the tool I called from the command line) has done almost all the work for me, I just had to compile the code and call it’s methods, then I did some reading on Axis’s site and decided to experiment with direct usage of Axis.

First I set up a simple SOAP service, using Axis’s ability to build a service from a java file, a very simple one:

Here is the code (helloEverybody.jws):

```
public class helloEverybody {

   public String hello() {
       String results = "Hello everybody";
     return results;
   }
}
```

I placed the file in c:\axis-1_1\webapps\axis and started looking at it’s WSDL description with the url http://localhost:8080/axis/helloEverybody.jws?wsdl

Looking at it you can quickly get some infos:

The service section:

```
<wsdl:service name="helloEverybodyService">
<wsdl:port binding="impl:helloEverybodySoapBinding" name="helloEverybody">
<wsdlsoap:address location="http://localhost:8080/axis/helloEverybody.jws"/>
</wsdl:port>
```

```
</wsdl:service>
```

Shows you the service address (to be called by your client) and the soap port, named “helloEverybody”.

```
<wsdl:operation name="hello">
<wsdlsoap:operation soapAction=""/>
        <wsdl:input name="helloRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://DefaultNamespace" use="encoded"/>
</wsdl:input>
        <wsdl:output name="helloResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://localhost:8080/axis/helloEverybody.jws" use="encoded"/>
</wsdl:output>
</wsdl:operation>
```

And for details on the response:

```
<wsdl:message name="helloResponse">
<wsdl:part name="helloReturn" type="xsd:string"/>
</wsdl:message>
```

The rest of the WSDL shows clearly that there is only one method to be called “hello” which returns a response of type “string” (Axis’s site details mapping between xsd types and java types).

This is the jsp client I put up:

```
<%@ page contentType="text/html; charset=iso-8859-1" language="java" import="org.apache.axis.client.Service, org.apache.axis.encoding.XMLType, javax.xml.rpc.*;" errorPage="" %>
<html>
<head>
<title>SOAP client direct call</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body>
<%
   String host = "http://localhost:8080";
   String servicepath = "/axis/helloEverybody.jws";
   String endpoint = host   + servicepath;
   String method = "hello" ;
   String ret = null;
   Service service = new Service();
   Call call = (Call) service.createCall();
   call.setTargetEndpointAddress(endpoint);
   call.setOperationName(new javax.xml.namespace.QName (method));
   ret = (String) call.invoke(( Object [] )null);
   out.write("Got result : " + ret);
%>
</body>
</html>
```

The “endpoint” and “method” corresponds to what I wrote above.

According to Axis guides I created a service and a call from that service, then I set an endpoint and an operation name for that call. After this I invoked the call (without input parameters and displayed the response converting it to a string for compatibility.

Next steps will be securing the service with a Realm and https and consuming it from jsp with direct usage of Axis.