# Web services with Apache Axis

## 1. Installation

Installation instructions for Axis are available at http://ws.apache.org/axis/java/install.html
Here is a brief summary.

### 1.1. Application Server

Axis runs in an application server, *Tomcat* can be chosen if you haven't got one yet.
Download a 5.x version from http://tomcat.apache.org/ (we used the 5.5.12 version).
Just unzip it in a directory. In order to run Tomcat you just need to set up your `CLASSPATH` variable correctly.

### 1.2. Axis

Download Axis from http://ws.apache.org/axis/java/releases.html (we used the latest version 1.3).
"Un-targz" (or "un-zip") it in a temporary directory. Copy the directory named *axis* and located in *axis-1.3/webapps/* in the *tomcat_root_dir/webapps/* directory. So now you must have a *tomcat_root_dir/webapps/axis/WEB-INF/classes* directory and a *tomcat_root_dir/webapps/axis/WEB-INF/lib* one.The *lib* directory contains the libraries needed to use *Axis*, and you will have to put your *Java* classes implementing your services in the *classes* directory.

Now verify that *Axis* is up and running by looking at the web page : http://127.0.0.1:8080/axis
By clicking on the *Validation* link, you can check that all libraries needed were found. We recommend that you also download the two optional libraries, as they allow us to use file transfer abilities of web services.
You need to restart the application server in order that it is able to find the new libraries. Don't forget to add theses libraries to your `CLASSPATH` variable.
e.g. :

```
export CATALINA_HOME=/opt/apache-tomcat-5.5.12
export AXIS_HOME=$CATALINA_HOME/webapps/axis
export AXIS_LIB=$AXIS_HOME/WEB-INF/lib
export AXISCLASSPATH=$AXIS_LIB/axis.jar:$AXIS_LIB/commons-discovery-0.2.jar:.........
```

If after restarting the server some libraries remained not found, try copying its in the */opt/apache-tomcat-5.5.12/common/lib/* directory and restart the server once again.

### 1.3. Hello World WS

*Axis* offers two tools for generating :

- *WSDL* files from *Java* interfaces describing the services we want to implement.
- All the *Java* classes needed to run a service and writing a client from a *WSDL* file.

#### 1.3.1. Writing the interface

```
public interface HelloWorld extends java.rmi.Remote {
    public String getHelloWorld() throws java.rmi.RemoteException;
}
```

### 1.3.2. Generating the WSDL file

Compile the interface above and then use the *Java2WSDL* tool :

```
java org.apache.axis.wsdl.Java2WSDL -o helloworld.wsdl
-l"http://localhost:8080/axis/services/HelloWorld" -n"urn:helloworld"
-p"helloworld" "urn:helloworld" HelloWorld


-o : name of the file we want to create
-l : where we want the service to be reached by clients
-n : namespace used in the wsdl file generated
-p : put files in a helloworld package
```

This command create a wsdl file named *helloworld.wsdl*.

### 1.3.3. Generating the WS code

Then we can use the *WSDL2Java* in order to generate the skeleton implementation of our service :

```
java org.apache.axis.wsdl.WSDL2Java -o . -s helloworld.wsdl


-o : where we want to put the output files
-s : will make the tool generate the server side classes. By default, it only
generates the client utility classes.
```

### 1.3.4. Implementing and deploying the service

You just need to edit the **`HelloWorldSoapBindingImpl.java`** file. That's the file you have to modify to implement your service. Replace the **`return null`** statement by a **`return("Hello World")`**.

Then, compile all the classes and put the package with compiled classes in the *classes* directory indicated above.

Move in this directory containing the compiled classes, and type the `deploy` command (the `undeploy` command can be used in the same way) :

```
alias deploy='java org.apache.axis.client.AdminClient -p8080 deploy.wsdd'


-p : specify on which port the service will be available (We guess here your
application server is running on 8080 port)
```

Then you can check service has been well deployed by visiting the http://127.0.0.1/axis/service page.

### 1.3.5. Client tools

Classes generated by the *WSDL2Java* tool allow to code a client in a quick way.

Here is an example of a client calling the previous service, coded with the generated classes of *WSDL2Java.*

```
import helloworld.*;


public class ClientHelloWorld {
          public static void main(String[] args) {
                    try {
                              HelloWorldServiceLocator locator = new
HelloWorldServiceLocator();
                              HelloWorld service = locator.getgetHelloWorld();

System.out.println((String)service.getHelloWorld());
                    } catch (Exception e) {
                              System.err.println(e.getMessage());
                    }
          }
}
```