

Axis Installation

1. Edit your `.bash_profile` or `.cshrc` file to include the following lines at the **end of file**:

Bash Shell Users:

```
export AXIS_HOME=/netbin/axis-1_4
export AXIS_LIB=$AXIS_HOME/lib
export AXIS_CLASSPATH=$AXIS_LIB/axis-ant.jar:$AXIS_LIB/axis.jar:$AXIS_LIB/commons-
discovery-0.2.jar:$AXIS_LIB/commons-logging-1.0.4.jar:$AXIS_LIB/jaxrpc.jar:$AXIS_LIB/log4j-
1.2.8.jar:$AXIS_LIB/saaj.jar:$AXIS_LIB/wsdl4j-1.5.1.jar:$AXIS_LIB/activation.jar:$AXIS_LIB/
mailapi.jar:$AXIS_LIB/xmlsec-1.3.0.jar:
export CLASSPATH=$AXIS_CLASSPATH:$CLASSPATH
```

Source `~/.bash_profile` file by executing `source ~/.bash_profile` (instead, you can logout and login again).

C Shell Users:

```
setenv AXIS_HOME /netbin/axis-1_4
setenv AXIS_LIB ${AXIS_HOME}/lib
setenv AXIS_CLASSPATH ${AXIS_LIB}/axis-ant.jar:${AXIS_LIB}/axis.jar:${AXIS_LIB}/commons-
discovery-0.2.jar:${AXIS_LIB}/commons-logging-1.0.4.jar:${AXIS_LIB}/jaxrpc.jar:${AXIS_LIB}/log4j-
1.2.8.jar:${AXIS_LIB}/saaj.jar:${AXIS_LIB}/wsdl4j-1.5.1.jar:${AXIS_LIB}/activation.jar:${AXIS_LIB}/
mailapi.jar:${AXIS_LIB}/xmlsec-1.3.0.jar:
setenv CLASSPATH ${AXIS_CLASSPATH}:${CLASSPATH}
```

Source `~/.cshrc` file by executing `source ~/.cshrc` (instead, you can logout and login again).

NOTE: There is no newline in the line: `export AXIS_CLASSPATH ...` all the contents till `\".` must be on one line.

2. Go to `$CATALINA_HOME/webapps` and copy entire content of `axis` directory into `$AXIS_HOME/webapps/axis` using the following command:
`cp -r $AXIS_HOME/webapps/axis $CATALINA_HOME/webapps/`
3. (Re)Start tomcat web server using the following command:
`$CATALINA_HOME/bin/startup.sh`
4. Test if Axis is being run by the web server by pointing the browser to `http://HOSTNAME:PORTNUM/axis/` (where `HOSTNAME` and `PORTNUM` are the assigned webserver hostname and port number, respectively). You should see a validation page.
5. To deploy a sample webservice, change to `$AXIS_HOME/samples/stock` directory and execute the following command:

```
java org.apache.axis.client.AdminClient -lhttp://localhost:<PORTNUM>/axis/services/AdminService
  deploy.wsdd
```

If you open the URL `http://HOSTNAME:PORTNUM/axis` in a browser and click of List you should see this service listed.

6. To test the webservice, change to `$CATALINA_HOME/webapps/axis/WEB-INF/classes` directory and type:

```
java samples.stock.GetQuote IBM  
-lhttp://localhost:<PORTNUM>/axis/servlet/AxisServlet -uuser1 -wpass1 XXX
```

You should get the IBM stock value as the answer.

7. To undeploy the webservice, change to `$AXIS_HOME/samples/stock` directory and execute the following command:

```
java org.apache.axis.client.AdminClient  
-lhttp://localhost:<PORTNUM>/axis/services/AdminService undeploy.wsdd
```

Google Web Service

1. Create a directory where all client files are going to be kept for this service (e.g., `~/cs631/homework4/google`).
2. Change into previously created directory and use `GoogleSearch.wsdl` to create client side java file used by GoogleSearch web service by executing:

```
java org.apache.axis.wsdl.WSDL2Java http://api.google.com/GoogleSearch.wsdl
```

This will create a directory `GoogleSearch` with all of the stub classes in it.

3. Create a java client file `GoogleClient.java` to use the service in the same directory as all the stub classes were created (e.g., `~/cs631/homework4/google/GoogleSearch/GoogleClient.java`)

The code for this file is given in the Appendix A.

4. Change to the directory where all the Java source files were created (e.g., `~/cs631/homework4/google/GoogleSearch`) and compile all of the java files:

```
javac *.java
```

5. Test the service by executing the client from the directory `~/cs631/homework4/google/`:
`java GoogleSearch.GoogleClient`

Deploying your own Web Service

The following instructions illustrate how to deploy a new web service using Axis. There instructions are for deploying a simple application that does not use any packages or multiple class files. This example should be sufficient for completing the Homework. In case you are interesting in deploying a more complicated service follow the example provided at:

<http://www.onjava.com/pub/a/onjava/2002/06/05/axis.html?page=2>.

1. Consider a simple temperature conversion example, the service and client program are shown below:

```
public class TempConvert {

    public double getFahrenheit(double c) {
        return (9.0/5.0)*c + 32.0;
    }

    public double getCelsius(double f) {
        return (5.0/9.0)*(f - 32.0);
    }
}

public class TempConvertClient
{
    public static void main( String args[] ) throws Exception
    {
        TempConvert myTemp = new TempConvert();

        System.out.println("100 degree Celsius = " +
            myTemp.getFahrenheit(100.0) +
            " degree Fahrenheit");
        System.out.println("100 degree Fahrenheit = " +
            myTemp.getCelsius(100.0) +
            " degree Celsius");
    }
}
```

Compile and test the program. Copy the file TempConvert.java to \$CATALINA_HOME/webapps/axis and rename the file TempConvert.java as TempConvert.jws. You can do that using the following command:

```
cp TempConvert.java $CATALINA_HOME/webapps/axis/TempConvert.jws
```

You should have the Apache Tomcat server running or start the server before following the next steps. You can check if the service was deployed by connecting to the URL: <http://HOSTNAME:PORTNUM/axis/TempConvert.jws>.

2. In a separate directory generate the client side interfaces and write a client program to test the service. You can generate the client side interfaces using the following command (all in one line, here vulcan8 and port 8080 is used, you have to use the appropriate hostname and portnumber):

```
java org.apache.axis.wsdl.WSDL2Java http://vulcan8.cis.uab.edu:8080/axis/  
TempConvert.jws?wsdl
```

3. The above command will generate the interfaces in a directory `edu/uab/cis/vulcanX/TempConver_jws/`, where `vulcanX` corresponds to one of the Vulcan machines. Here is the client program to test the service:

```
import edu.uab.cis.vulcan8.axis.TempConvert_jws.*;

public class TempConvertClient
{
    public static void main( String args[] ) throws Exception
    {
        TempConvertService service = new TempConvertServiceLocator();
        TempConvert myTemp = service.getTempConvert();
        // call instance methods
        System.out.println("100 degree Celsius = " +
            myTemp.getFahrenheit(100.0) +
            " degree Fahrenheit");
        System.out.println("100 degree Fahrenheit = " +
            myTemp.getCelsius(100.0) +
            " degree Celsius");
    }
}
```

4. You can compile the client using **javac TempConvertClient.java** and run the program as **java TempConvertClient**, you should see the following output:

```
100 degree Celsius = 212.0 degree Fahrenheit
100 degree Fahrenheit = 37.77777777777778 degree Celsius
```

Appendix A

```
package GoogleSearch;

public class GoogleClient {

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

        // create service instance
        GoogleSearchService service = new GoogleSearchServiceLocator();
        GoogleSearchPort google = service.getGoogleSearchPort();

        String key = "9X117vdQFHITuWswp1YRBTZcjjMdPATY";
        String query = "UAB CIS";
        int start = 0;
        int maxResults = 10;
        boolean filter = true;
        String restrict = "";
        boolean safeSearch = true;
        String languageRestrict = "lang_en";
        String inputEncoding = "UTF-8";
        String outputEncoding = "UTF-8";

        GoogleSearchResult gsr =
            google.doGoogleSearch(
                key,
                query,
                start,
                maxResults,
                filter,
                restrict,
                safeSearch,
                languageRestrict,
                inputEncoding,
                outputEncoding);

        ResultElement[] re = gsr.getResultElements();

        for (int i = 0; i < re.length; i++) {
            System.out.println(re[i].getURL());
        }
    }
}
```