

Developing Web Services Using Apache Axis2 Eclipse Plugins - Part 1

By *sandakith*Created *2007-06-10 19:11*

The tutorial is written for Eclipse SDK v3.2 and Axis2 Eclipse Plugin v1.3.

Contents

- [Introduction](#) ^[1]
- [Tutorial Scenario](#) ^[2]
- [Getting Started](#) ^[3]
- [Apache Axis2 Service Archive Generator Wizard ? Eclipse Plugin](#) ^[4]
- [Apache Axis2 Code Generator Wizard - Eclipse Plugin](#) ^[5]
- [Part 1 - Axis2 Eclipse Plugin demonstration of Bottom Up Approach of Web services Development.](#) ^[6]
- [Part 2 - Axis2 Eclipse Plugin demonstration of Top Down Approach of Web services Development.](#) ^[7]
- [Conclusion](#) ^[8]
- [References](#) ^[9]
- [See Also](#) ^[10]

Introduction



[Apache Axis2](#) ^[11] is the most popular and widely used core engine for Web services. It is a complete re-design and re-write of the widely used [Apache Axis](#) ^[12] SOAP stack built on the lessons learned from Apache Axis. Apache Axis2 is more efficient, more modular, and more XML-oriented than the older version. It is carefully designed to support the easy addition of plugin "modules" that extend its functionality for features such as security and reliability. On the other hand, Eclipse is a project aiming to provide a universal tool set for development. It is an open source IDE that is mostly provided in Java, but the development language is independent and not limited in any way. It is the most widely used IDE for most computer languages, especially for JAVA.

^[13]

Developing applications using any programming language is becoming easier with the availability of tooling. If tooling is available in areas like JAVA application development, it will facilitate faster and easier application development, and also increase the productivity of the developers. Most integrated development environments (IDEs) address the tooling and the features available around a particular area of development. Most IDEs available today go beyond supplying traditional tooling facilities, which address

only certain areas in programming. They are becoming **frameworks** for developing applications.

This tutorial mainly focuses on the two above mentioned tools available for Web service developers. They are, Axis2 Eclipse Plugins, which are built around Eclipse-the Framework available for JAVA application development, and Axis2- the widely used core engine for Web services. These tools help the developers to easily expose their available plain old JAVA applications as Web services, using wizards.

This tutorial is split into two parts covering two basic Web services scenarios of Web service in the Bottom Up (Code First) and Top Down (Contract First) approaches with the Axis2 Eclipse Plugins.

PART 1 - Axis2 Eclipse Plugin demonstration of Bottom Up Approach of Web services Development.

PART 2 - Axis2 Eclipse Plugin demonstration of Top Down Approach of Web services Development.

Tutorial Scenario

Assume that you are a Java developer who wants to expose your available application as a Web service. This tutorial covers developing, deploying, and testing a temperature conversion application as a Web service in the Top Down (Contract First) and Bottom Up (Code First) approaches. In Web services terminology, Bottom Up (Code First) is used where the developer starts with the business logic, which is the code, and then develop and deploy the code as a Web service. The Top Down (Contract First) approach starts from the Web service descriptions, which is the WSDL, and then goes on to expose the Web service.

Only two Eclipse plugins are used in the process. The tutorial uses the Axis2 Web application deployed in your servlet container as the Web service deployment engine. Also, it is assumed that you know the basics of how to use Eclipse as your Java development environment, and therefore it does not cover areas such as creating a JAVA project in an Eclipse workspace and compiling and building a JAVA project in Eclipse.

Getting Started

These are the tools used in the tutorial.

- 1) Java Development Kit 1.4.2.x ([Download](#) ^[14])
- 2) Eclipse SDK 3.2.x ([Download](#) ^[15])
- 3) Apache Tomcat 4.1.x ([Download](#) ^[16])
- 4) Axis2 Web Application ([Download](#) ^[17])
- 5) Axis2 Eclipse Plugins ([Download](#) ^[18])

Now let us focus on the two plugins mentioned in the tutorial scenario.

1) Apache Axis2 Service Archive Generator Wizard ? Eclipse Plugin

As a part of the Axis2 tool set, the service archive generator is an important tool that allows the generation of service archives (an "aar" file or a "jar" file) that can be deployed as a Web service to Axis2.

2) Apache Axis2 Code Generator Wizard - Eclipse Plugin

The Axis2 Code Generator Wizard is the other important tool that allows you to generate WSDL2Java and Java2WSDL code, which is the heart of developing and testing Web services.

To start developing Web services, you need to download, install the two plugins, and verify that the plugins are working properly. The installation of plugins into Eclipse is as simple as downloading the plugins from the Apache Axis2 download page and extracting them to the Eclipse plugins directory. You can download the two plugins from the Axis2 Tools Page. Refer to the [installation instructions](#).^[18]

Other than the two plugins, we need the Axis2 runtime to deploy the developed Web services. We use the Axis2 Web Application which can be deployed in any popular servlet container. (You can download the Axis2 Web Application from the [Axis2 Download page](#).^[17]) You have to just place it in the repository of the servlet container and run it. For example, if you are using Apache Tomcat, just copy the downloaded .war file, put it in the webapp folder, and start the servlet container.

PART 1 - The Bottom Up Approach in Web Services Development Using the Apache Axis2 Eclipse Plugin

- Start Eclipse SDK. First we need to create a JAVA project in Eclipse. (Follow the instruction on Eclipse SDK help.) You can give any name to the JAVA project, but for clarity, let us create a JAVA project called ?TemperatureWebService?.

In the Bottom Up Approach, we start with the service implementation and then build the deployable Web service component. We will be using the TemperatureConverter class definition as the logic for developing, deploying, and testing the Web service.

- Create a custom package ws.example appropriately and include the TemperatureConverter.java file in that package. Compile and build the project.

```
package ws.example;
/**
 * Temperature Converter Implementation Class
 */
public class TemperatureConverter {
    /**
     * util method to convert celsius to fahrenheit
     * @param cValue : double value of celsius
     * @return calculated value of fahrenheit
     */
}
```

```
public double c2fConversion(double cValue) {
    return ((cValue * 9.0)/5.0 )+ 32.0;
}
/**
 * util method to convert fahrenheit to celsius
 * @param fValue : double value of fahrenheit
 * @return calculated value of celsius
 */
public double f2cConversion(double fValue) {
    return ((fValue - 32.0) * 5.0) / 9.0;
}}
```

- After successfully building the project, we will create the service archive using the Axis2 Eclipse Service Archiver Plugin. On the File menu, click New and then click Other to access the Axis2 wizards. Else, you can press Ctrl+N.

Note : At any given time, you can go back through the wizards, change the settings, and even start the wizards all over again to create a different Web service on another JAVA project available in your workspace.

- Select the Axis2 Service archiver and click **Next**. You will see the Axis2 service definition selection page.

On this page, select the output location of the ?TemperatureWebService? Java project that we previously developed. Here we point to the service implementation classes. If there is more than one class, you only have to point to the JAVA project build location. The wizard includes all the implementation files. To be on the safe side, if you are going to expose a complex project as a Web service, it's better to select the **include .class files** check box to omit unnecessary resource files that will increase the size of the deployable service archive that we are going to create.

- After selecting the correct output location, click **Next**.

On this page, you can browse for the WSDL file. If you do not want to add a WSDL file to the service archive, select the **Skip WSDL** check box. Else you can select the **Select WSDL** check box, and specify the location of the WSDL file. We will skip the WSDL for the moment.

- Click **Next**.

This page is to add the libraries. The library name (with the full path) can be specified by either typing it or browsing for it. Once the library name with the full path is entered, click **Add** to add the library to the list. The added libraries should be displayed in the Added libraries list. You can add as many external libraries as you wish.

If any added library has to be removed, select it from the Added libraries list and click **Remove**.

- For this example, we do not need any external libraries added to the generated service. Therefore, click **Next** to proceed to the next step.

This page is about the services.xml generation. If you have a custom services.xml file, you can select the services.xml file on this page by browsing for it or you can generate the service XML automatically. The browsing option is disabled when the **Generate service xml automatically** check box is selected.

- For this example, we use the automatically generated services.xml rather than a custom written one. Therefore, select the **Generate service xml automatically** check box, and then click **Next**.

Note that this page will appear only if you selected to generate the services.xml automatically in the previous step. (If you selected a services.xml file, then you will be directed to the last page of the wizard.) After entering the correct service name and a valid fully qualified class name, load the existing methods of that class by clicking **Load**. If it is successfully loaded, you will see a table at the bottom of the page with the details of the loaded class. You can specify the methods to include in the services.xml by selecting the corresponding check boxes.

- Select the **Search declared method only** check box, to remove the inherited methods from the class. We do not need the inherited methods to be exposed and our only interest is in temperature conversion logic. Click **Next** to proceed to the last page of the wizard.

On the last page of the wizard, specify the output file location and the output archive file name to

Testing the Temperature Converter Service

Now that we have the service up and running, let's go ahead and test the created Temperature Converter service. This consists of two steps.

We have to generate the client code and invoke the Web service. For that we use the Axis2 Eclipse Codegen Plugin to create a WSDL from a Java source. For demonstration purposes of the Eclipse Codegen Axis2 Eclipse Codegen Plugin we will do that in two steps. As the first step, we will create the WSDL, and then as the second step use that WSDL to generate code for the client.

Also note that you can skip the WSDL generation part since the deployed Web service is up and running. The WSDL will be generated when we click Temperature Converter under the available services in the Axis2 Web application. However, since the main aim of this tutorial is to introduce the Axis2 Eclipse Plugins, we are not going to use that option. We will use the Axis2 Eclipse Codegen Plugin to generate the WSDL for us.

Step 1 : Generate the WSDL from the Java source

- Start the Axis2 Eclipse Codegen Plugin by selecting it and clicking **Next** on the New wizard page.

We are going to create the WSDL using the Axis2 Eclipse Codegen Plugin Java2wsdl option.

- On the first page, select the **Generate a WSDL from a JAVA source and file** option. Then, click **Next**.

On this page, select the class to be exposed and the relevant .jar files /classes to be loaded as the classpath.

Add the folder location of the class files of our project. After specifying the fully qualified class name click on the **Add Folder** button and add the location of the class files of your project. After the libraries have been set, click **Test Class Loading** to test whether the class is loadable.

- Test the class loading by clicking **Test Class Loading**

Unless the class loading is successful, the **Next** button will not be enabled. As we did on the service archive generation, we have to enter the fully qualified class name as the service class. Then select the project output folder.

- Once the class loading is successful, click **Next**. The page below will appear.

This page allows the parameters to be modified by setting the options for the generator. **Note:** If you customize these parameters (instead of the default), you will be generating a different Web service descriptor with different parameters than what we have generated by using the service archive wizard.

NOTE : If you are using the 1.3 version of the Axis2 Eclipse Plugin you need to change the schema target namespace to `http://example.ws` to comply it with the Axis2 1.3 version. Please drop the trailing `/xsd` part of the schema target namespace. All other earlier version than 1.3 do not need this change.

- We will accept all the default values. Click **Next**.

- Here you can specify the output file location by typing or browsing for it using the **Browse** button. You have the option of browsing only Eclipse workspace projects by selecting the **Add the source to a project on current eclipse workspace** option. Else you have the option of saving the codegen results to the file system.
- Once the output file location and the output WSDL file name is added, click **Finish** to complete generation.



A message appears informing you that all the operations were completed successfully. You have successfully completed the Java2WSDL code generation and created the WSDL of the Temperature Converter Service.

Step 2 : Generate code and invoke the service

In this step, we have to generate code, which represents the client side stubs, using the Axis2 Eclipse Codegen Plugin and test the deployed Temperature Converter Service by using the WSDL file generated

in Step 1.

To create the client stub using the Axis2 Eclipse Codegen Plugin wsdl2java option:

- Start the Axis2 Eclipse Codegen Plugin by selecting it and clicking **Next** on the new wizard page.
- Create the client stub using the Axis2 Eclipse Codegen Plugin wsdl2java option by selecting the **Generate Java source code from WSDL file** option. Click **Next**.

- Select the previously generated WSDL location by browsing for it. Click **Next**.

Once the WSDL file is selected, the next page will take you to the page from where the codegen options are to be selected. By far this is the most important page in this wizard. This page determines the characteristics of the code being generated.

If this is the first time you invoked the wizard, you will see that the most common options are set by default. Advanced users will find it very easy to turn the knobs using these options. You can select Custom from the Select Codegen Options list and then change/edit the fields that you need. We are going to create stubs accepting the default values.

- Accept the default settings and click **Next**.

On the final page of the wizard, you can specify the output file path by typing or browsing for it using the **Browse** button. You have the option of browsing only Eclipse workspace projects by selecting the **Add the source to a project on current eclipse workspace** option. Else you have the option to save the codegen results to the file system.

- Click **Add the source to a project on current eclipse workspace** and select the project that you have created earlier.
- Select the **Add codegen jars to the codegen resulted project** check box so that we can easily compile the code without worrying about adding Axis2 libraries to the JAVA project classpath.

- Click **Finish**. A message appears informing you that you have successfully completed the WSDL2Java code generation.



- The client stub files will be generated in the project.

In order to compile the code we need to add the generated libraries to the project library path. You

can add the .jar files in the lib directory by navigating the project properties of the Java project.

After adding the required libraries, you will be able to clean build the project without any errors.

- Now we will write the Client, and use this client to invoke the Web service deployed earlier. Add the following TemperatureConverterServiceClient.java class to the project,

```
package ws.example;
public class TemperatureConverterServiceClient {
    public static void main(String[] args) {
        TemperatureConverterStub stub;
        try {
            double c_value = 32;
            stub = new TemperatureConverterStub
("http://localhost:8080/axis2/services/TemperatureConverter");
            TemperatureConverterStub.C2FConversion c2f = new TemperatureConv
c2f.setCValue(c_value);
            TemperatureConverterStub.C2FConversionResponse res = stub.c2FCo
System.out.println("C Value : "+c_value+ "\tResult : " +res.get_
TemperatureConverterStub.F2CConversion f2c = new TemperatureConv
f2c.setFValue(res.get_return());
            TemperatureConverterStub.F2CConversionResponse res1 = stub.f2CCo
System.out.println("F Value : "+res.get_return()+ "\tResult : "
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

NOTE : If you are using an early version of the Axis2 Eclipse Plugins than 1.3, you may have to change the client according to the stubs that have been generated. Previous versions of Axis2 did generate multiple stubs for each port type of the wsdl and you may use the example client given below for those versions.

```
package ws.example;
import java.rmi.RemoteException;
import org.apache.axis2.AxisFault;
public class TemperatureConverterServiceClient {
    public static void main(String[] args) {
```

```

TemperatureConverterTemperatureConverterSOAP11PortStub stub;
try {
    double c_value = 32;
    stub = new TemperatureConverterTemperatureConverterSOAP11PortStub
("http://localhost:8080/axis2/services/TemperatureConverter");
    TemperatureConverterTemperatureConverterSOAP11PortStub.C2FConversion c2f
    = new TemperatureConverterTemperatureConverterSOAP11PortStub.C2FConversion(c_value);
    c2f.setCValue(c_value);
    TemperatureConverterTemperatureConverterSOAP11PortStub.C2FConversionResult res
    = stub.c2FConversion(c2f);
    System.out.println("C Value : "+c_value+ "\tResult : " +res.get_return());
    TemperatureConverterTemperatureConverterSOAP11PortStub.F2CConversion f2c
= new TemperatureConverterTemperatureConverterSOAP11PortStub.F2CConversion(res);
    f2c.setFValue(res.get_return());
    TemperatureConverterTemperatureConverterSOAP11PortStub.F2CConversionResult res2
= stub.f2CConversion(f2c);
    System.out.println("F Value : "+res.get_return()+ "\tResult : " +res2.get_return());
    } catch (AxisFault e) {
        e.printStackTrace();
    } catch (RemoteException e) {
        e.printStackTrace();
    }
}
}}

```

After adding the TemperatureConverterServiceClient.java class to the project, compile, and build the project. After that you can run the client to test the service. If you have successfully carried out all the steps, the temperature converter service will be invoked and the results of the service will be displayed on the command line output.

Note: The stub assumes that you run Tomcat on 8080 (if not, change the stub URL and re-run the client).

Sample Project Src Download

- [Without libraries](#) ^[19]
- [With libraries](#) ^[20]

Conclusion

Web service creation, deployment, and testing is no more a time consuming task. With the Axis2 Eclipse Plugins you can create, deploy, and test your Web services using a wizard. The time to write, deploy, and test a Web service is reduced to a minimum with the introduction of these tools.

[PART 2 - Top Down Approach](#) ^[21]

See also..

[How to create a web service using WSAS tools in 3 steps](#) ^[22]

[How to create a web service client using WSAS tools in 3 steps](#) ^[23]

[How to debug a web service using WSAS tools in 3 steps](#) ^[24]

[How to edit a web service while testing it using WSAS tools in 3 steps](#) ^[25]

References

Apache Axis2 - <http://ws.apache.org/axis2> ^[26]

Apache Axis2 Tools - <http://ws.apache.org/axis2/tools/index.html> [18]

Eclipse.org - <http://ws.apache.org/axis2/tools/index.html> [18]

Eclipse Plugins - <http://www.eclipseplugincentral.com/> [27]

Author

Lahiru Sandakith P.G. WSO2, Inc. sandakith at wso2 dot com

[Articles Introductory](#)

© 2010 WSO2 Inc.

Footer

- [Licenses](#)
- [Privacy Policy](#)
- [Terms of Use](#)
- [Community Guidelines](#)
 - [Feedback](#)
 - [wso2.com](#)

Source URL: <http://wso2.org/library/1719>

Links:

- [1] <http://wso2.org/library/1719#Introduction>
- [2] <http://wso2.org/library/1719#TuteScenario>
- [3] <http://wso2.org/library/1719#GetStart>
- [4] <http://wso2.org/library/1719#ServiceArvhiver>
- [5] <http://wso2.org/library/1719#CodeGenerator>
- [6] <http://wso2.org/library/1719#Part1>
- [7] <http://wso2.org/library/1719#Part2>
- [8] <http://wso2.org/library/1719#Conclusion>
- [9] <http://wso2.org/library/1719#References>
- [10] http://wso2.org/library/1719#see_also
- [11] <http://ws.apache.org/axis2/>
- [12] <http://ws.apache.org/axis/>
- [13] <http://wso2.com/products/create/wso2-web-services-application-server-wsas?libaddate=06082009>
- [14] <http://java.sun.com/j2se/1.4.2/download.html>
- [15] <http://www.eclipse.org/downloads/>
- [16] <http://tomcat.apache.org/tomcat-4.1-doc/index.html>
- [17] http://ws.apache.org/axis2/download/1_2/download.cgi
- [18] <http://ws.apache.org/axis2/tools/index.html>
- [19] <http://wso2.org/files/TemperatureWebServiceWithoutLibs.zip>
- [20] <http://wso2.org/files/TemperatureWebServiceWithLibs.zip>
- [21] <http://wso2.org/library/1986>
- [22] <http://wso2.org/library/tutorials/create-axis2-web-service-in-3-steps-using-eclipse>
- [23] <http://wso2.org/library/tutorials/creating-web-service-client-3-steps-using-eclipse>
- [24] <http://wso2.org/library/tutorials/debug-your-axis2-web-service-3-steps-using-eclipse>
- [25] <http://wso2.org/library/tutorials/live-edit-your-axis2-web-service-using-eclipse>
- [26] <http://ws.apache.org/axis2>
- [27] <http://www.eclipseplugincentral.com/>