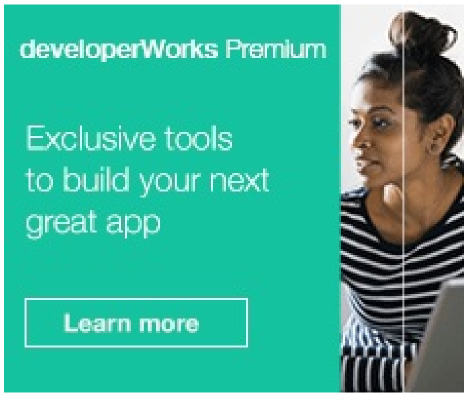
Web services using Apache Axis2

Objectives

The objective of this tutorial is to demonstrate you how to create and invoke a web service using Apache Axis2.

- Hosting a simple web service using Axis2
- Invoke the web service over HTTP using Axis2



Example code

To demonstrate the above objectives, I've used the simple "*StockQuote*" example which comes with standard Axis2 binary distribution and we will use only one operation "*getStockQuote*" for simplicity.

Prerequisites

I assume that you are a Java programmer and you are conceptually aware of web services. I have tried my best to present this tutorial in a very intuitive fashion so that you can follow along visually without downloading the example code or search any missing steps. I've used only open source tools or software in this entire attempt and I will try to make it simple so that even a newbie will be able to try out all the use cases in this tutorial.

Software required

The development and testing of all the artifacts are done on Windows XP SP3. If you need to try this on any other platform, please replace with the platform specific software and commands.

Back to top

Build and invoke a web service using Apache Axis2

WSDL file

Let us start with the WSDL file we are going to use throughout this tutorial. We will save it as *StockQuoteService.wsdl*.

Listing 1. StockQuoteService.wsdl

```
<wsdl:definitions xmlns:axis2="http://quickstart.samples/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:ns="http://quickstart.samples/xsd"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://quickstart.samples/">
<wsdl:types>
```

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://quickstart.samples/xsd">
<xs:element name="getPrice">
<xs:complexType>
<xs:sequence>
<xs:element name="symbol" nillable="true" type="xs:string" />
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="getPriceResponse">
<xs:complexType>
<xs:sequence>
<xs:element name="return" nillable="true" type="xs:double" />
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
</wsdl:types>
<wsdl:message name="getPriceMessage">
<wsdl:part name="part1" element="ns:getPrice" />
</wsdl:message>
<wsdl:message name="getPriceResponseMessage">
<wsdl:part name="part1" element="ns:getPriceResponse" />
</wsdl:message>
<wsdl:portType name="StockQuoteServicePortType">
<wsdl:operation name="getPrice">
<wsdl:input message="axis2:getPriceMessage" />
<wsdl:output message="axis2:getPriceResponseMessage" />
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="StockQuoteServiceSOAP11Binding"
type="axis2:StockQuoteServicePortType">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document" />
<wsdl:operation name="getPrice">
<soap:operation soapAction="urn:getPrice" style="document" />
<wsdl:input>
<soap:body use="literal" namespace="http://quickstart.samples/"/>
</wsdl:input>
<wsdl:output>
<soap:body use="literal" namespace="http://quickstart.samples/"/>
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="StockQuoteService">
<wsdl:port name="StockQuoteServiceSOAP11port"
    binding="axis2:StockQuoteServiceSOAP11Binding">
<soap:address
location="http://localhost:8080/axis2/services/StockQuoteService"/>
```

```
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

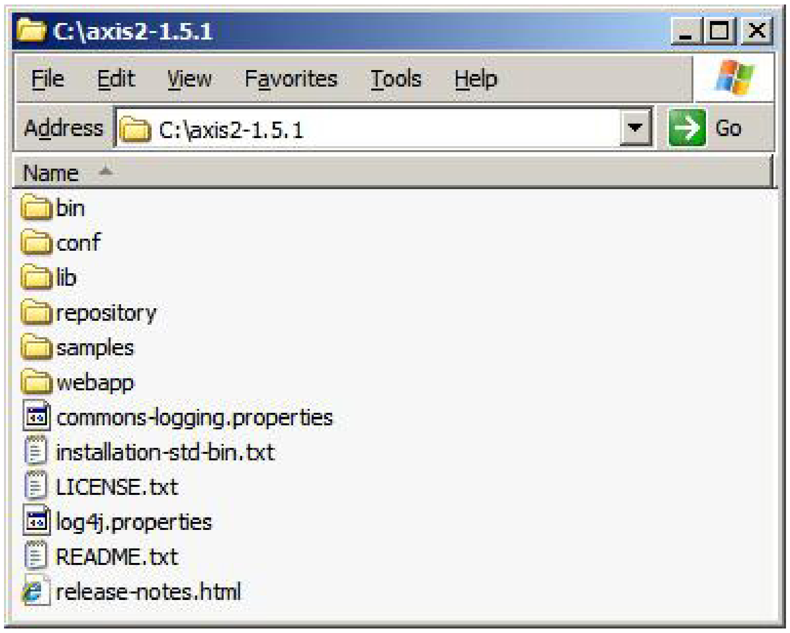
If you closely read this WSDL, we can see that the service name is *StockQuoteService* and it has a single operation *getPrice* which takes a string input and returns a *double* value.

Creating service components from the WSDL file

To create the web service server-side components from a WSDL file, there are quite a number of ways available. Of which the most common one is to create a Dynamic Web Project in Eclipse J2EE version and generate web service skeleton from the WSDL using eclipse's in-built web service support plugins. This we will discuss after talking about the manual way of creating web services. But to understand more about the Axis2's service creation feature, we will go with the more configurable way of creating service using Axis2's command line tools. This involves a few more steps than how eclipse does, but it is better to learn how it is done. The details are explained in the steps below:

1. Create the folder structure *C:\StockQuoteWS* (you are free to create any folder structure). But for simplicity, throughout this tutorial we will be using the above folder structure.
2. In this directory, create a new file (*StockQuoteService.wsdl*) and paste the above WSDL contents to this file.
3. Download the Apache Axis2 binary distribution from the site provided in the prerequisites section of this tutorial. Unzip the *axis2-1.5.1-bin.zip* to C:\ drive. This will create a folder *axis2-1.5.1*. The folder structure should look like Figure 1 below.

Figure 1. Axis2-1.5.1-bin.zip extracted

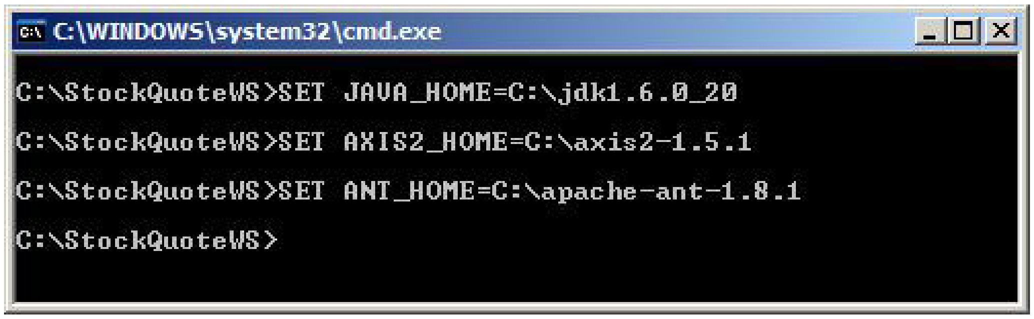


bin
conf
lib
repository
samples
webapp
commons-logging.properties
installation-std-bin.txt
LICENSE.txt
log4j.properties
README.txt
release-notes.html

4. Similarly, unzip *apache-ant-1.8.1-bin.zip* to C:\ drive. This will create a folder *apache-ant-1.8.1* with all the Apache Ant components.
5. Open a command prompt window (using the shortcut *Start* > *Run* > *cmd*) and change the directory to *C:\StockQuoteWS*

6. Before we build anything with Axis2, we have to take care of little house keeping. We need to set the *AXIS2_HOME, ANT_HOME and JAVA_HOME* environment variables. These variables can be set as system variables or just specific to this particular command prompt window.
7. Type in the following 3 commands in the command prompt (enter one after the other) to set the environment variables. Note: You may have to change the value of the variables if you have Axis2, Ant or JDK installed in different paths other than the ones specified.
   1. `SET JAVA_HOME=C:\jdk1.6.0_20`
   2. `SET AXIS2_HOME=C:\axis2-1.5.1`
   3. `SET ANT_HOME=C:\apache-ant-1.8.1`

Figure 2. Set environment variables

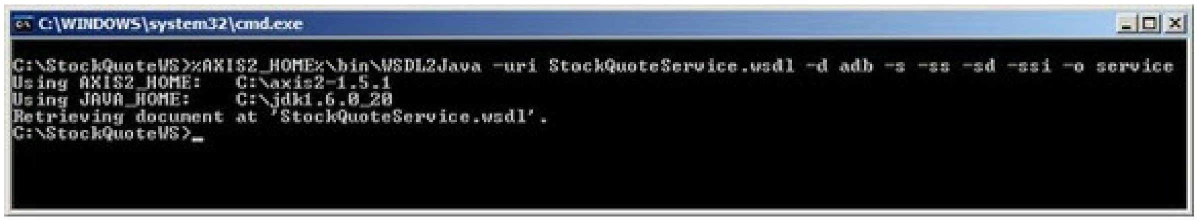


```
C:\StockQuoteWS>SET JAVA_HOME=C:\jdk1.6.0_20

C:\StockQuoteWS>SET AXIS2_HOME=C:\axis2-1.5.1

C:\StockQuoteWS>SET ANT_HOME=C:\apache-ant-1.8.1

C:\StockQuoteWS>
```

8. We will generate the service components for the web service. For this we will use the standard Axis2's `WSDL2Java` command. We will use Axis2 Data binding Framework (ADB) as the data binding framework (you are free to choose any data binding framework which Axis2 supports). For generating the server-side components of the web service, execute the following command in the same command prompt window.

   ```
   %AXIS2_HOME%\bin\WSDL2Java -uri StockQuoteService.wsdl -d adb -s -ss -sd -ssi -
   o service
   ```

Figure 3. Set environment variables



```
C:\StockQuoteWS>%AXIS2_HOME%\bin\WSDL2Java -uri StockQuoteService.wsdl -d adb -s -ss -sd -ssi -o service
Using AXIS2_HOME:   C:\axis2-1.5.1
Using JAVA_HOME:    C:\jdk1.6.0_20
Retrieving document at 'StockQuoteService.wsdl'.
C:\StockQuoteWS>_
```

The options include:

Table 1. WSDL2Java options

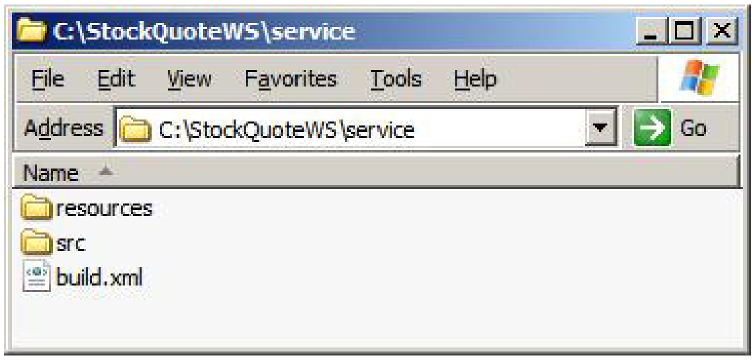
Option syntax	Description
`-uri <url or path>`	A url or path to a WSDL

Option syntax	Description
`-d <databinding>`	Databinding framework to be used. Valid databinding(s) are adb, xmlbeans, jibx and jaxbri (Default: adb)
`-s`	Generate sync style code only (Default: off)
`-ss`	Informs the Axis2 WSDL2Java tool to generate server side code. Generate server side code (i.e. skeletons) (Default: off)
`-sd`	Generate service descriptor (i.e. services.xml). (Default: off). Valid with -ss
`-ssi`	Generate an interface for the service implementation (Default: off)
`-o <path>`	Specify a directory path for the generated code

There are other options also available with the `WSDL2Java` tool and can be seen by just invoking `%AXIS2_HOME%\bin\WSDL2Java` command without any options.

9. This will create the service components and the below folders and files. `WSDL2Java` command creates a folder named service and generates the components inside this directory.

Figure 4. Folders generated from WSDL2Java command

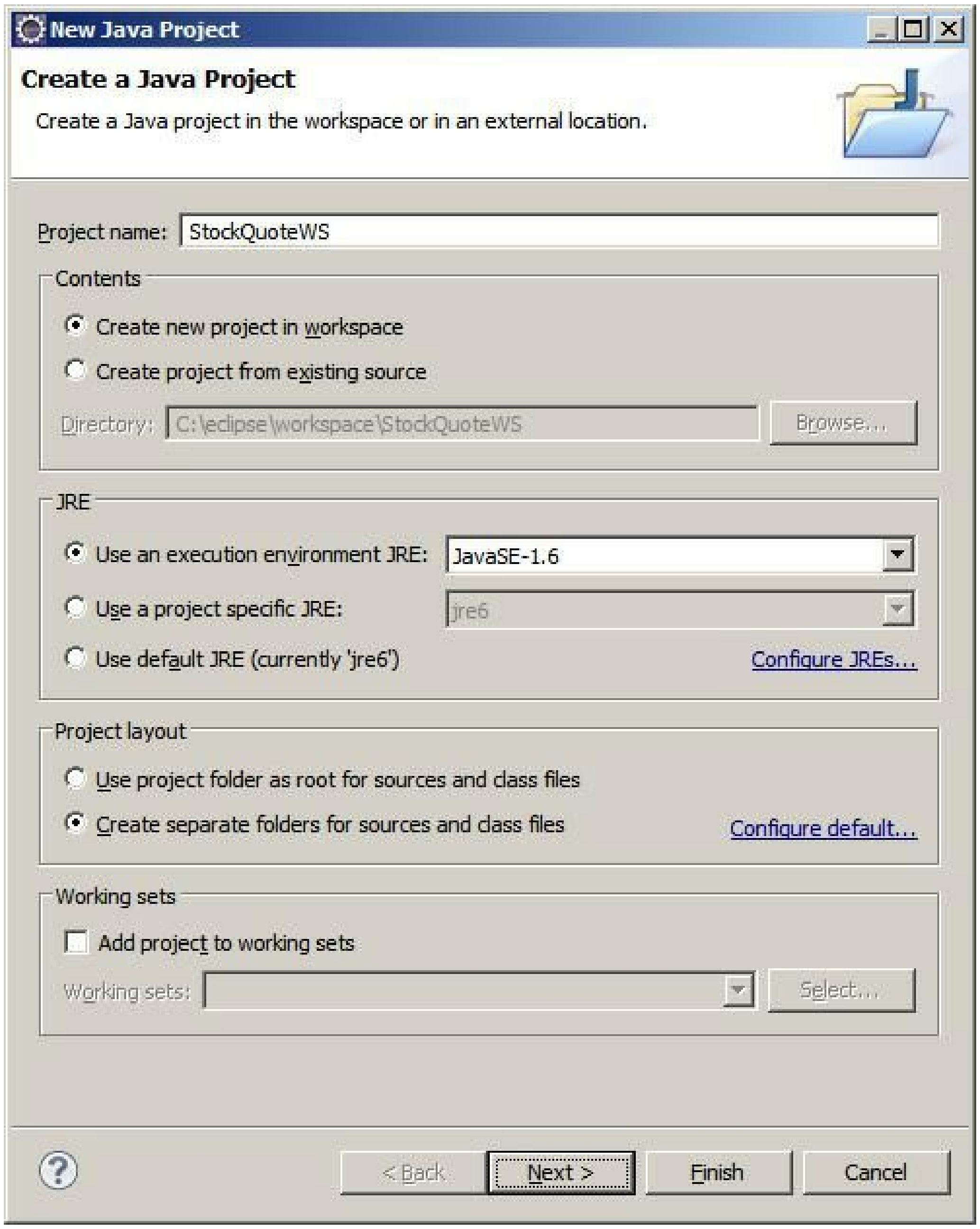


We are completed the creation of the service components, next we will complete the service implementation. Axis2 generates only the skeleton template for us and we need to provide implementation of how our web service should behave. Which means, we have to write java code for the service implementation; what all actions the service have to perform. Just like there are several ways creating web service components, there are different ways of completing the service implementation. If you have used Eclipse for skeleton generation from WSDL file, the same project would have been having the generated java files waiting for the implementation to be completed. As we have generated the service components using Axis2 command line tool, we need to follow a different path of action. How we can complete the service implementation is explained in the below section.

Completing the web service implementation

1. Create a new Java project in Eclipse with name **StockQuoteWS**.

Figure 5. Create Java Project in Eclipse IDE


New Java Project
Create a Java Project
Create a Java project in the workspace or in an external location.

Project name: StockQuoteWS

Contents
- (●) Create new project in workspace
- ( ) Create project from existing source

Directory: C:\eclipse\workspace\StockQuoteWS

JRE
- (●) Use an execution environment JRE: JavaSE-1.6
- ( ) Use a project specific JRE: jre6
- ( ) Use default JRE (currently 'jre6')

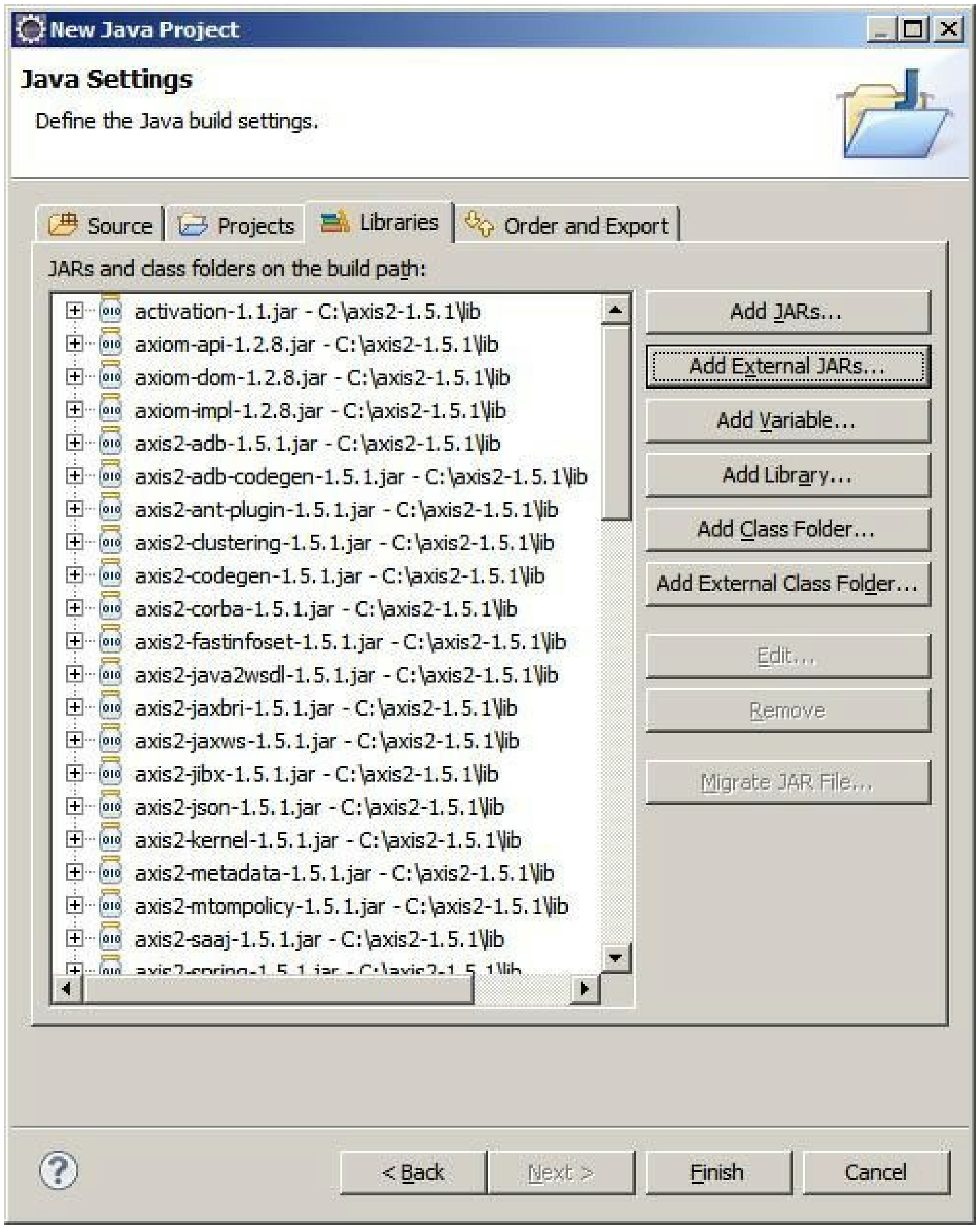
Project layout
- ( ) Use project folder as root for sources and class files
- (●) Create separate folders for sources and class files

Working sets
- [ ] Add project to working sets

Working sets:

2. Add the jar files in Axis2 lib folder to the Libraries of this project (by using Add External JARs option).

Figure 6. Project library settings



```
activation-1.1.jar - C:\axis2-1.5.1\lib
axiom-api-1.2.8.jar - C:\axis2-1.5.1\lib
axiom-dom-1.2.8.jar - C:\axis2-1.5.1\lib
axiom-impl-1.2.8.jar - C:\axis2-1.5.1\lib
axis2-adb-1.5.1.jar - C:\axis2-1.5.1\lib
axis2-adb-codegen-1.5.1.jar - C:\axis2-1.5.1\lib
axis2-ant-plugin-1.5.1.jar - C:\axis2-1.5.1\lib
axis2-clustering-1.5.1.jar - C:\axis2-1.5.1\lib
axis2-codegen-1.5.1.jar - C:\axis2-1.5.1\lib
axis2-corba-1.5.1.jar - C:\axis2-1.5.1\lib
axis2-fastinfoset-1.5.1.jar - C:\axis2-1.5.1\lib
axis2-java2wsdl-1.5.1.jar - C:\axis2-1.5.1\lib
axis2-jaxbri-1.5.1.jar - C:\axis2-1.5.1\lib
axis2-jaxws-1.5.1.jar - C:\axis2-1.5.1\lib
axis2-jibx-1.5.1.jar - C:\axis2-1.5.1\lib
axis2-json-1.5.1.jar - C:\axis2-1.5.1\lib
axis2-kernel-1.5.1.jar - C:\axis2-1.5.1\lib
axis2-metadata-1.5.1.jar - C:\axis2-1.5.1\lib
axis2-mtompolicy-1.5.1.jar - C:\axis2-1.5.1\lib
axis2-saaj-1.5.1.jar - C:\axis2-1.5.1\lib
axis2-spring-1.5.1.jar - C:\axis2-1.5.1\lib
```

3. Copy the generated service components (all the java files from folder *C:\StockQuoteWS\service\src*) which Axis2 has generated to the source folder (src) of this project.
4. The new Java project should look like Figure 7 below.

Figure 7. Package Explorer view in Eclipse

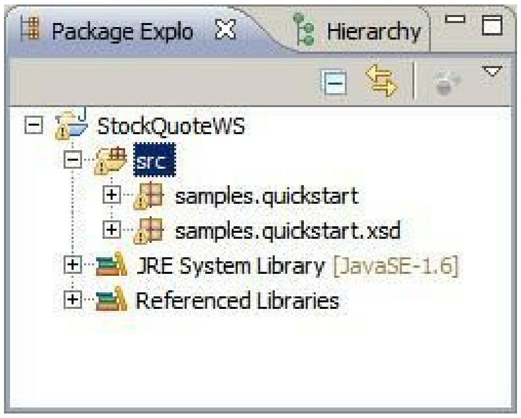
5. Open the class *samples.quickstart.StockQuoteServiceSkeleton* for editing and remove the line

```
throw new java.lang.UnsupportedOperationException("Please implement " +
this.getClass().getName() + "#getPrice");
```

Add the below 3 lines in place of the deleted line.

```
samples.quickstart.xsd.GetPriceResponse response = new
samples.quickstart.xsd.GetPriceResponse();
```

```
response.set_return(100.00d);
return response;
```



The entire class file should look like:

```
/**
*StockQuoteServiceSkeleton.java
*This file was auto-generated from WSDL
*by the Apache Axis2 version: 1.5.1 Built on :
*Oct 19, 2009 (10:59:00 EDT)
*/

package samples.quickstart;

/**
*    StockQuoteServiceSkeleton java skeleton for the axisService
*/

public class StockQuoteServiceSkeleton implements
StockQuoteServiceSkeletonInterface {

/**
*   Auto generated method signature
*       @param getPrice0
*/
public samples.quickstart.xsd.GetPriceResponse
    getPrice(samples.quickstart.xsd.GetPrice getPrice0) {
    samples.quickstart.xsd.GetPriceResponse response = new
    samples.quickstart.xsd.GetPriceResponse();
    response.set_return(100.00d);
    return response;
    }
}
```

From the auto generated method signature, we can make out that the operation *getPrice* should return an object of type *samples.quickstart.xsd.GetPriceResponse*. So we will first create an object of this type. Next we will set the return value. Axis2 provides a method *set_return* to set the return value. So in this case, we will set the value *100.00d* as our operation has to return a *double* value.

The service implementation is now complete. For keeping the implementation simple, we will just return a value of 100.00 irrespective of the input symbol (symbol is the input of the operation *getPrice*, refer to the WSDL). You can provide a different implementation for the operation, but it should return an object of type *samples.quickstart.xsd.GetPriceResponse*.

6. Copy all the files from the source folder (*src* folder) of this java project to *C:\StockQuoteWS\service\src* folder (maintain the same package structure). Actually we are replacing the generated files in folder *C:\StockQuoteWS\service\src* with the ones in the eclipse project. Just replacing the *C:\StockQuoteWS\service\src\samples\quickstart\StockQuoteServiceSkeleton.java* in the generated files with the one from Eclipse project will also be fine.

We are finished with the service implementation. The next step is the creation of the web service archive to be deployed on Axis2.

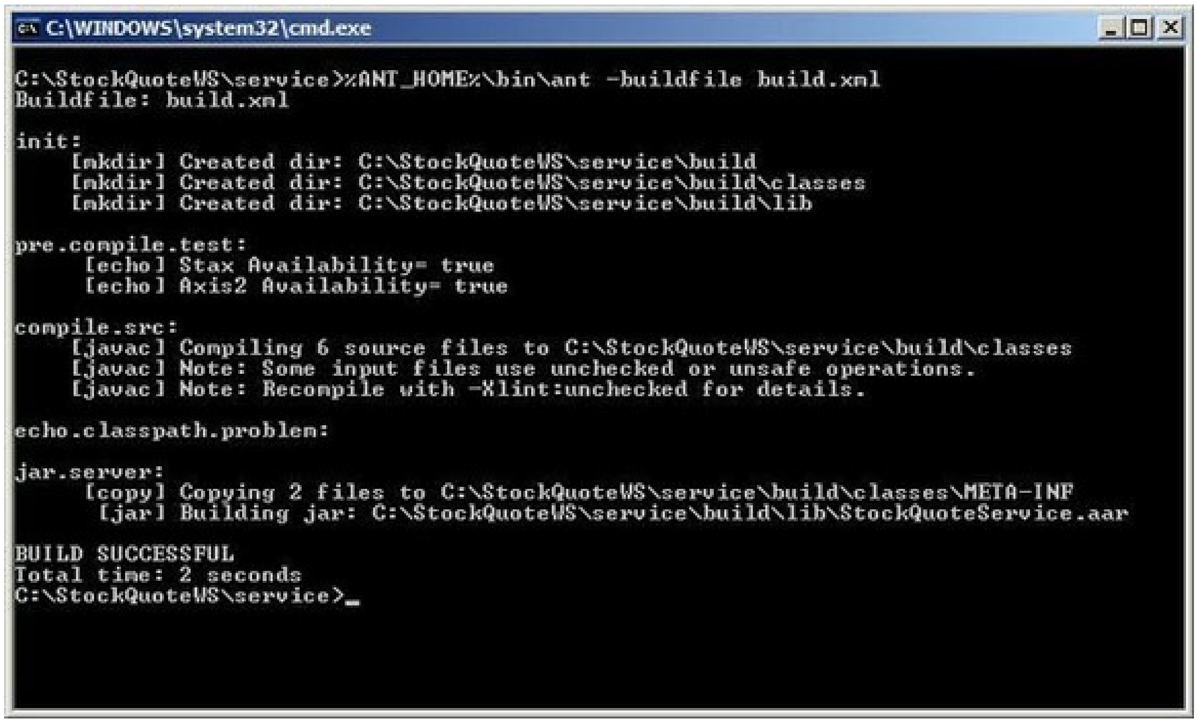
Creation of web service archive

Just like we create war or ear file in the case of web and enterprise applications, web service deployment on a stand-alone Axis2 server is also commonly (not mandatory always) done with the use of an archive file. This archive file is called an "*aar*" (Axis2 Archive) and has a "*.aar*" extension in the file name.

Web service archive creation is very simple in Axis2. For this also Axis2 has generated an artifact (a *build.xml* file in folder *C:\StockQuoteWS\service*). This artifact is nothing but an Ant build file. We will use Apache Ant to run this file. The default task of this build file will create the web service archive and keep it in *build\lib* folder. In the earlier command prompt window change to directory *C:\StockQuoteWS\service* and run the following command to create the web service archive.

```
%ANT_HOME%\bin\ant -buildfile build.xml
```

Figure 8. Output window of ANT build



```
C:\StockQuoteWS\service>%ANT_HOME%\bin\ant -buildfile build.xml
Buildfile: build.xml

init:
    [mkdir] Created dir: C:\StockQuoteWS\service\build
    [mkdir] Created dir: C:\StockQuoteWS\service\build\classes
    [mkdir] Created dir: C:\StockQuoteWS\service\build\lib

pre.compile.test:
     [echo] Stax Availability= true
     [echo] Axis2 Availability= true

compile.src:
    [javac] Compiling 6 source files to C:\StockQuoteWS\service\build\classes
    [javac] Note: Some input files use unchecked or unsafe operations.
    [javac] Note: Recompile with -Xlint:unchecked for details.

echo.classpath.problem:

jar.server:
     [copy] Copying 2 files to C:\StockQuoteWS\service\build\classes\META-INF
      [jar] Building jar: C:\StockQuoteWS\service\build\lib\StockQuoteService.aar

BUILD SUCCESSFUL
Total time: 2 seconds
C:\StockQuoteWS\service>_
```

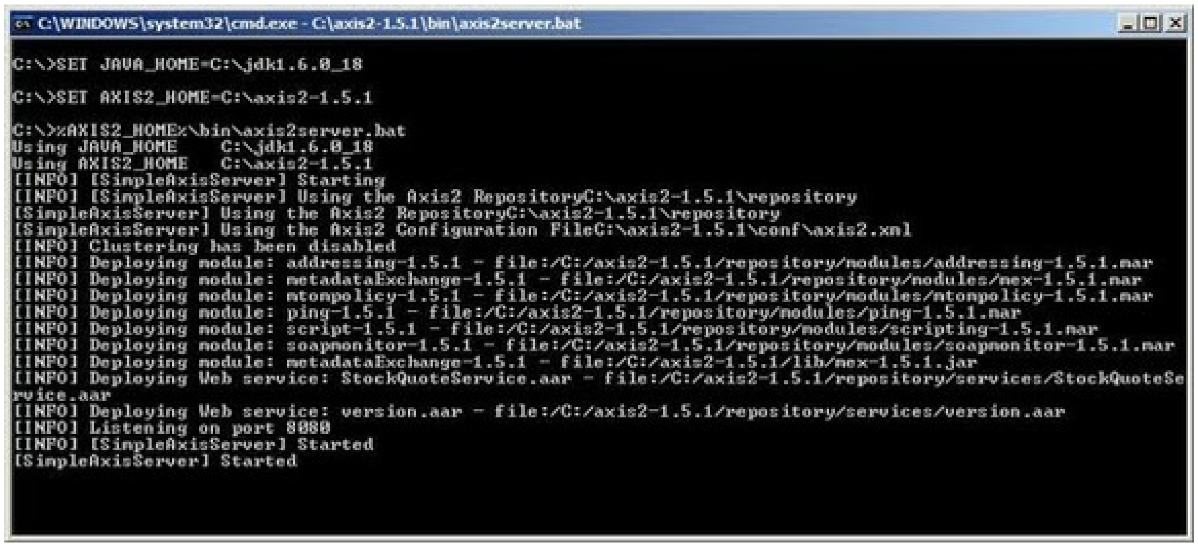
This will create *StockQuoteService.aar* file in folder *C:\StockQuoteWS\service\build\lib*. Our web service is now ready for deployment.

Deployment of web service in Axis2

Deployment of web service is quite simple in Axis2. All you have to do is to copy the web service archive to *AXIS2_HOME\repository\services* folder. So here we have to copy *C:\StockQuoteWS\service\build\lib\StockQuoteService.aar* to *C:\axis2-1.5.1\repository\services* folder. To complete the deployment and to check whether it has been deployed or not, we need to start the Axis2 server. Open a new command prompt and run the following 3 commands (one after the other).

```
SET JAVA_HOME=C:\jdk1.6.0_18
SET AXIS2_HOME=C:\axis2-1.5.1
%AXIS2_HOME%\bin\axis2server.bat
```

Figure 9. Starting Axis2 server



C:\WINDOWS\system32\cmd.exe - C:\axis2-1.5.1\bin\axis2server.bat

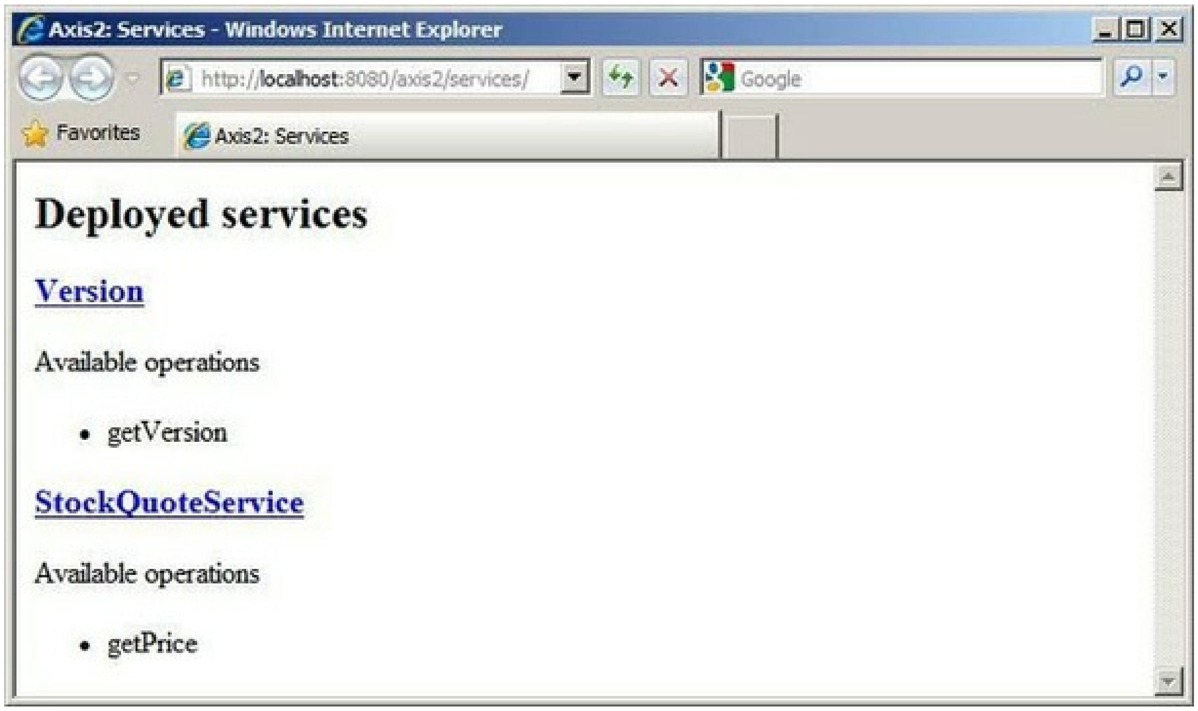
```
C:\>SET JAVA_HOME=C:\jdk1.6.0_18

C:\>SET AXIS2_HOME=C:\axis2-1.5.1

C:\>%AXIS2_HOME%\bin\axis2server.bat
Using JAVA_HOME    C:\jdk1.6.0_18
Using AXIS2_HOME   C:\axis2-1.5.1
[INFO] [SimpleAxisServer] Starting
[INFO] [SimpleAxisServer] Using the Axis2 RepositoryC:\axis2-1.5.1\repository
[SimpleAxisServer] Using the Axis2 RepositoryC:\axis2-1.5.1\repository
[SimpleAxisServer] Using the Axis2 Configuration FileC:\axis2-1.5.1\conf\axis2.xml
[INFO] Clustering has been disabled
[INFO] Deploying module: addressing-1.5.1 - file:/C:/axis2-1.5.1/repository/modules/addressing-1.5.1.mar
[INFO] Deploying module: metadataExchange-1.5.1 - file:/C:/axis2-1.5.1/repository/modules/mex-1.5.1.mar
[INFO] Deploying module: mtompolicy-1.5.1 - file:/C:/axis2-1.5.1/repository/modules/mtompolicy-1.5.1.mar
[INFO] Deploying module: ping-1.5.1 - file:/C:/axis2-1.5.1/repository/modules/ping-1.5.1.mar
[INFO] Deploying module: script-1.5.1 - file:/C:/axis2-1.5.1/repository/modules/scripting-1.5.1.mar
[INFO] Deploying module: soapmonitor-1.5.1 - file:/C:/axis2-1.5.1/repository/modules/soapmonitor-1.5.1.mar
[INFO] Deploying module: metadataExchange-1.5.1 - file:/C:/axis2-1.5.1/lib/mex-1.5.1.jar
[INFO] Deploying Web service: StockQuoteService.aar - file:/C:/axis2-1.5.1/repository/services/StockQuoteSe
rvice.aar
[INFO] Deploying Web service: version.aar - file:/C:/axis2-1.5.1/repository/services/version.aar
[INFO] Listening on port 8080
[INFO] [SimpleAxisServer] Started
[SimpleAxisServer] Started
```

From the output, we can make out that the Axis2 server is running on port 8080 (`[INFO] Listening on port 8080`).

(To change this default port, edit the *AXIS2_HOME\conf\axis2.xml*. This can done by changing the value of parameter port (`<parameter name="port">8080</parameter>`) under `<transportReceiver name="http" class="org.apache.axis2.transport.http.SimpleHTTPServer">` section in *axis2.xml*).

Open a browser and open the address http://localhost:8080/axis2.

Figure 10. Deployed services on Axis2



Axis2: Services - Windows Internet Explorer
http://localhost:8080/axis2/services/
Google
Favorites
Axis2: Services

Deployed services

Version

Available operations

- getVersion

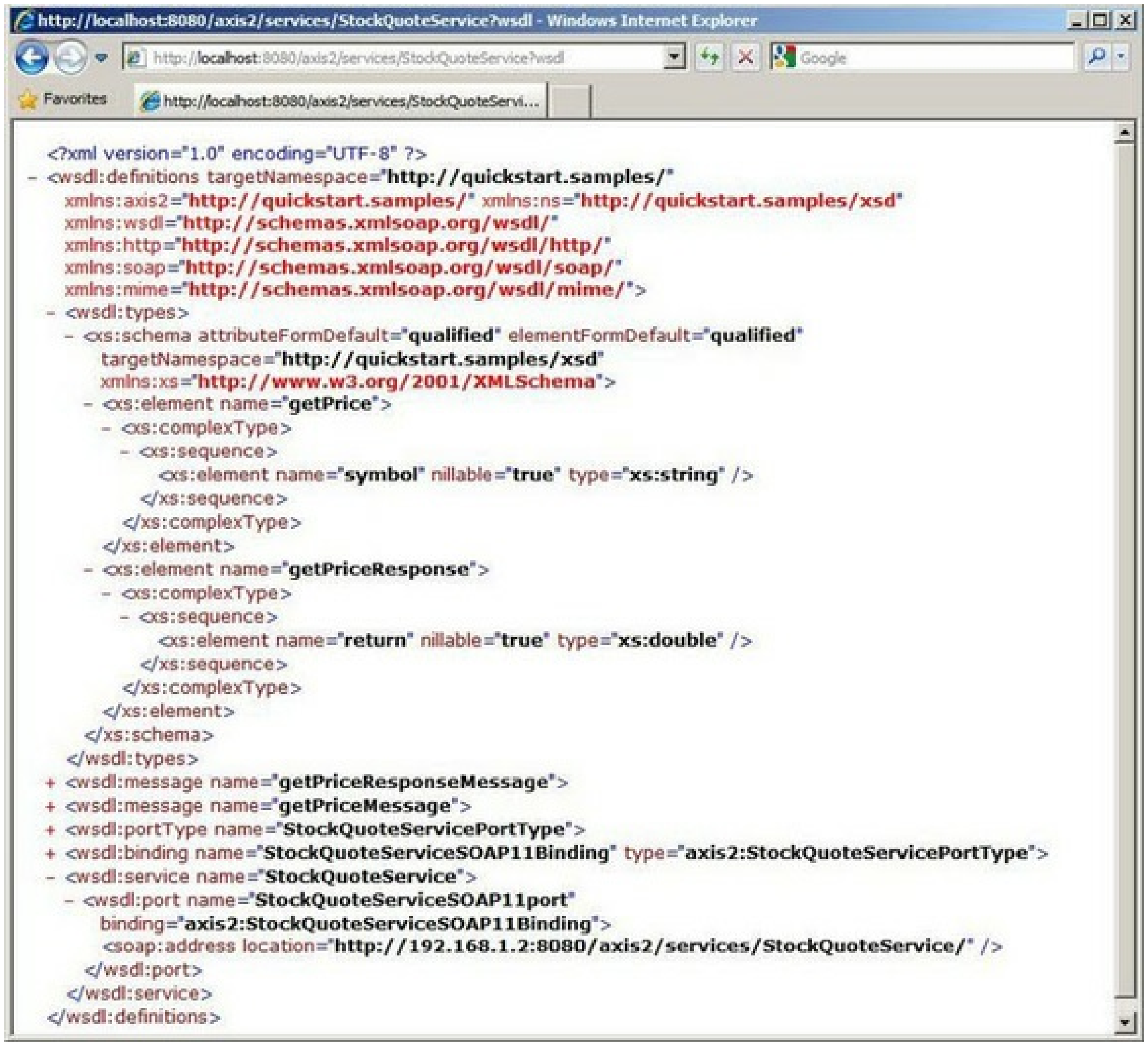
StockQuoteService

Available operations

- getPrice

This will list the web services which are deployed. Click on the *StockQuoteService* link to view the WSDL of the service.

Figure 11. Web service WSDL



http://localhost:8080/axis2/services/StockQuoteService?wsdl - Windows Internet Explorer
Favorites

```
<?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions targetNamespace="http://quickstart.samples/"
    xmlns:axis2="http://quickstart.samples/" xmlns:ns="http://quickstart.samples/xsd"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/">
  - <wsdl:types>
    - <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified"
        targetNamespace="http://quickstart.samples/xsd"
        xmlns:xs="http://www.w3.org/2001/XMLSchema">
      - <xs:element name="getPrice">
        - <xs:complexType>
          - <xs:sequence>
              <xs:element name="symbol" nillable="true" type="xs:string" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      - <xs:element name="getPriceResponse">
        - <xs:complexType>
          - <xs:sequence>
              <xs:element name="return" nillable="true" type="xs:double" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:schema>
    </wsdl:types>
  + <wsdl:message name="getPriceResponseMessage">
  + <wsdl:message name="getPriceMessage">
  + <wsdl:portType name="StockQuoteServicePortType">
  + <wsdl:binding name="StockQuoteServiceSOAP11Binding" type="axis2:StockQuoteServicePortType">
  - <wsdl:service name="StockQuoteService">
    - <wsdl:port name="StockQuoteServiceSOAP11port"
        binding="axis2:StockQuoteServiceSOAP11Binding">
        <soap:address location="http://192.168.1.2:8080/axis2/services/StockQuoteService/" />
      </wsdl:port>
    </wsdl:service>
  </wsdl:definitions>
```

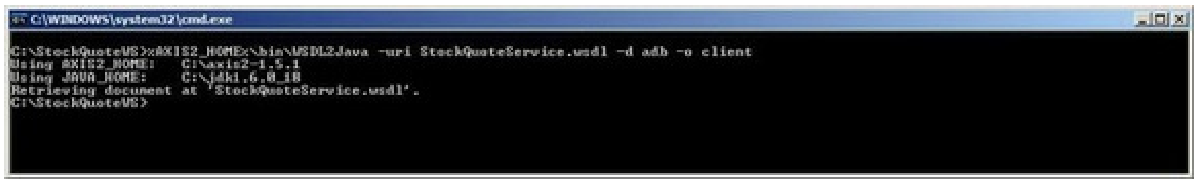
The web service is ready to for service. Next we will learn how we can invoke this web service.

Web service invocation

To get the service of a web service, we have to invoke it. There are plenty of ways to invoke a web service. Right from, opening a plain socket connection to the web service and passing the SOAP input, to creating dynamic clients, there are many approaches to invoke a web service. It is up to the implementation which method to use to invoke a web service. In our tutorial, we will generate the client of the deployed web service and invoke the web service using the client stubs. For this we will use the standard Axis2's WSDL2Java command. The command for creating service components and clients command are the same but with different arguments. We will use Axis2 Data binding Framework (ADB) as the data binding framework (you are free to choose any data binding framework which Axis2 supports) for the client. For generating the client, execute the following command in the earlier command prompt window.

```
%AXIS2_HOME%\bin\WSDL2Java -uri StockQuoteService.wsdl -d adb -o client
```

Figure 12. Client generation in Axis2

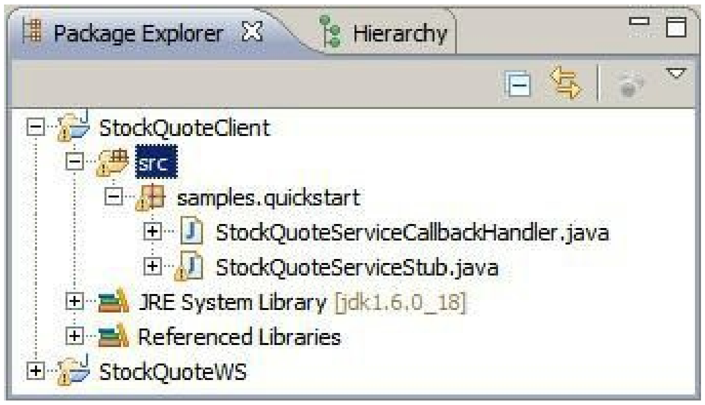


```
C:\StockQuoteWS>%AXIS2_HOME%\bin\WSDL2Java -uri StockQuoteService.wsdl -d adb -o client
Using AXIS2_HOME:   C:\axis2-1.5.1
Using JAVA_HOME:    C:\jdk1.6.0_18
Retrieving document at 'StockQuoteService.wsdl'.
C:\StockQuoteWS>
```

Similarly as we have created the *StockQuoteWS* java project in eclipse, we need to create another java project named **StockQuoteClient** and add all the jars in *AXIS2_HOME/lib* folder to the classpath of this project. Copy the generated client components (all the java files from folder *C:\StockQuoteWS\client\src* which Axis2 has generated) to the source folder (*src*) of this project.

The new Java project should look like Figure 13 below.

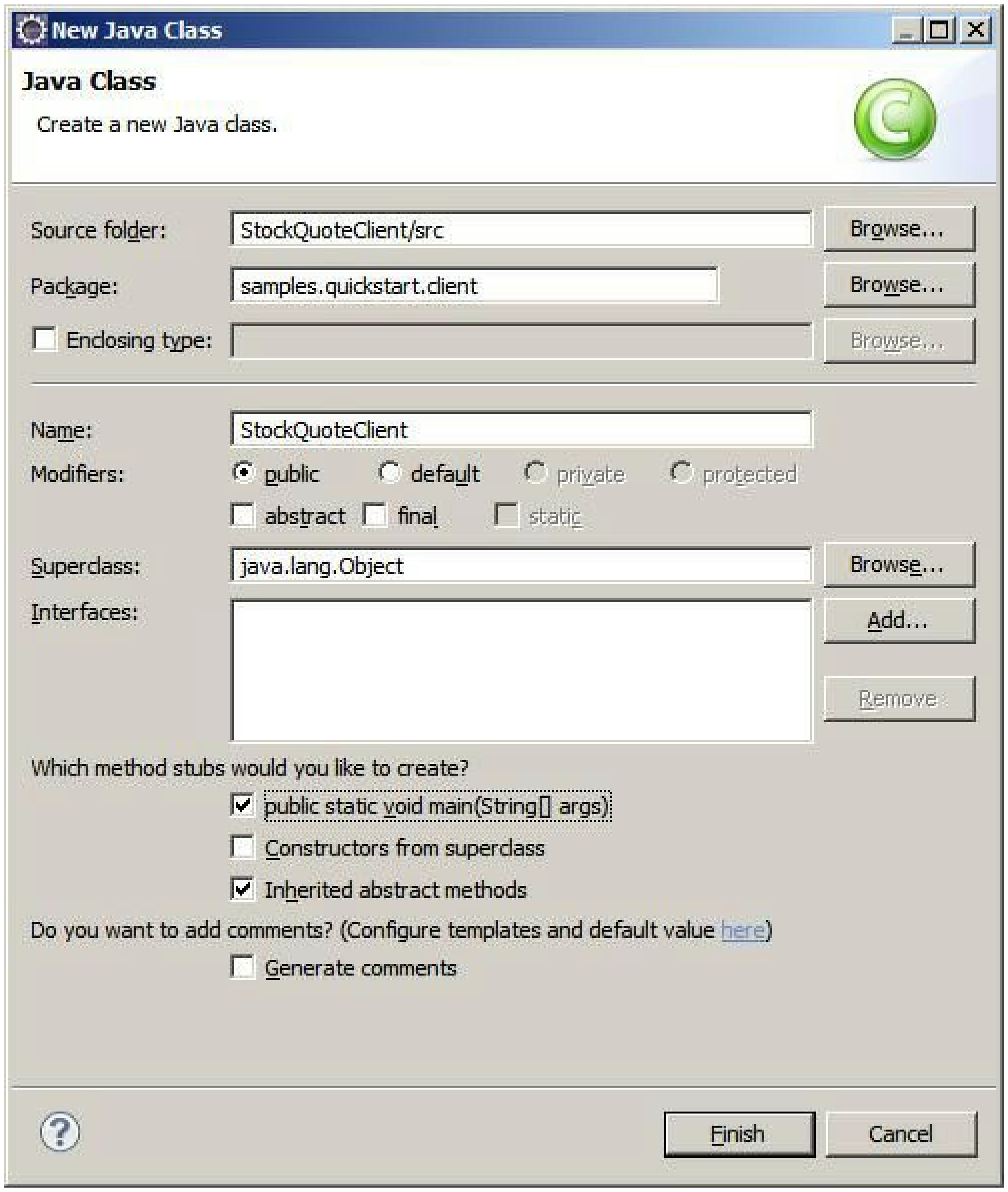
Figure 13. Eclipse Package Explorer view for web service client



Package Explorer
Hierarchy
StockQuoteClient
src
samples.quickstart
StockQuoteServiceCallbackHandler.java
StockQuoteServiceStub.java
JRE System Library [jdk1.6.0_18]
Referenced Libraries
StockQuoteWS

Next create a new class in this project named *samples.quickstart.client.StockQuoteClient* and add a main method to it.

Figure 14. Create New stub class


New Java Class

Java Class

Create a new Java class.

Source folder: StockQuoteClient/src

Package: samples.quickstart.client

☐ Enclosing type:

Name: StockQuoteClient

Modifiers: ● public ○ default ○ private ○ protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☑ public static void main(String[] args)

☐ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

The new class should look like the one below:

```
package samples.quickstart.client;
public class StockQuoteClient {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
    }
}
```

We will invoke the web service using the web service stub components. For that we need to add the few lines of code inside the main method. The final class should look like the one below:

```
package samples.quickstart.client;

public class StockQuoteClient {
    public static void main(String[] args) {
        try {
   samples.quickstart.StockQuoteServiceStub stub =
                    new samples.quickstart.StockQuoteServiceStub();
   samples.quickstart.StockQuoteServiceStub.GetPrice request =
                    new samples.quickstart.StockQuoteServiceStub.GetPrice();
   request.setSymbol("ABCD");
   samples.quickstart.StockQuoteServiceStub.GetPriceResponse
                  response =
   stub.getPrice(request);
   System.out.println(response.get_return());
        } catch (org.apache.axis2.AxisFault
   e) {
   e.printStackTrace();
        } catch (java.rmi.RemoteException
   e) {
   e.printStackTrace();
        }
    }
}
```

Here, first we create an object of the *samples.quickstart.StockQuoteServiceStub* which will be used to invoke the service skeleton using Axis2 APIs. If you open the *StockQuoteServiceStub.java* class, you can see how it is done in more detail (as it is not in the scope of this tutorial, we will not discuss it here). Then we create the input for the method *getPrice* and call the *getPrice* method of the stub. This will internally make a web service call to the *StockQuoteService* and get the response.

In the above code listing, we create an object of the *samples.quickstart.StockQuoteServiceStub* class. A stub is piece of software which is used by the client application to invoke a web service (mainly used for invoking web service’s operation). Then we create an object (*samples.quickstart.StockQuoteServiceStub.GetPrice*) of the input to the web service. Then we invoke the operation *getPrice* on the stub and it will return an object of *samples.quickstart.StockQuoteServiceStub.GetPriceResponse* on successful invocation.

After running the program, you can see *100.0* in the output. Remember, this is the value we set in the service implementation.

The same client can be used to invoke the *StockQuote* web service, with a minor change, if you deploy your service on a different machine/server. For this all you have to change is the way you create the *StockQuoteServiceStub*. For invoking the same service running on different target endpoint address, we will create the *StockQuoteServiceStub* by passing the target endpoint address to the constructor of *StockQuoteServiceStub*. For example:

```
samples.quickstart.StockQuoteServiceStub stub =
    new samples.quickstart.StockQuoteServiceStub
       ("http://<server-name or server-ip>:<axis2-listen-port>
        /axis2/services/StockQuoteService");
```

The rest of the client code will remain the same for invoking this web service (if the service name, operations and input and output remains the same).

Back to top

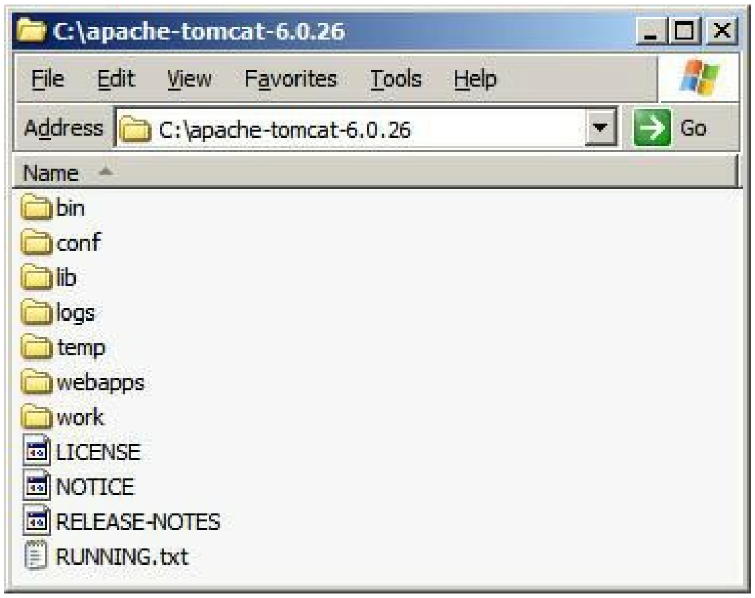
Creating service components from the WSDL file (using Eclipse IDE)

Before start with creating the service components from the WSDL file using Eclipse IDE, we need to setup *Apache Tomcat 6*.

Apache Tomcat installation

Extract *apache-tomcat-6.0.26.zip* to the C:\ drive. This will create a directory *C:\apache-tomcat-6.0.26*.

Figure 15. apache-tomcat-6.0.26.zip extracted

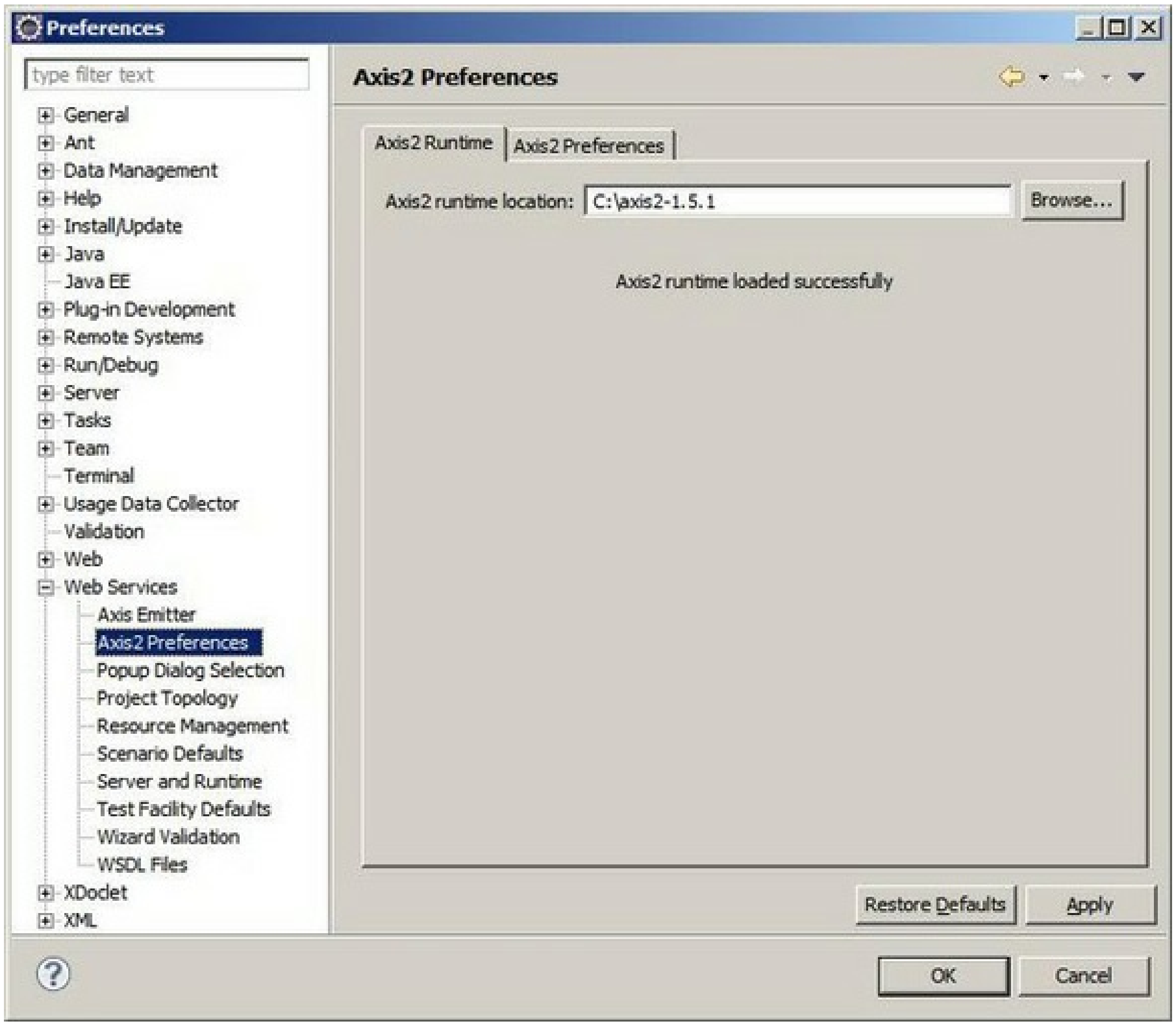

C:\apache-tomcat-6.0.26
File Edit View Favorites Tools Help
Address C:\apache-tomcat-6.0.26 Go
Name

bin
conf
lib
logs
temp
webapps
work
LICENSE
NOTICE
RELEASE-NOTES
RUNNING.txt

Eclipse IDE settings

For creating web service implementation using Eclipse IDE, just like we set the environment variables (*AXIS2_HOME etc.*), we need to inform Eclipse the path of Axis2. For this, open Eclipse IDE; go to *Window* > *Preferences* in the Eclipse IDE's menu. Expand the *Web Services* section and select *Axis2 Preferences*. On the right-hand side, browse and select the *Axis2 runtime location*. You should get the *Axis2 runtime loaded successfully* message.

Figure 16. Axis2 preferences in Eclipse

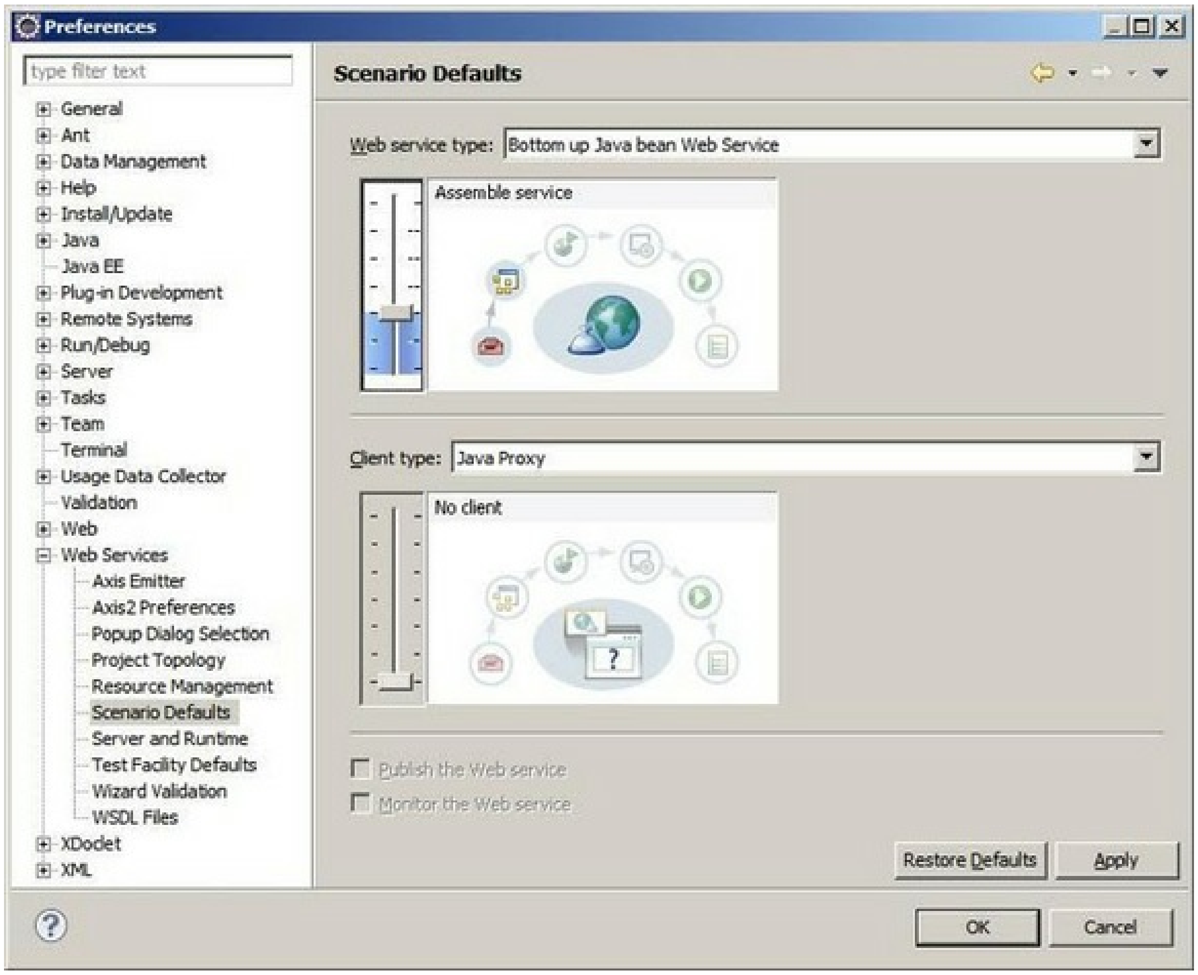


Axis2 runtime location: C:\axis2-1.5.1

Axis2 runtime loaded successfully

Then select *Scenario Defaults*. From this window, drag the first slider down till you see *Assemble service*.

Figure 17. Axis2 preferences(Scenario Defaults) in Eclipse



Web service type: Bottom up Java bean Web Service

Assemble service

Client type: Java Proxy

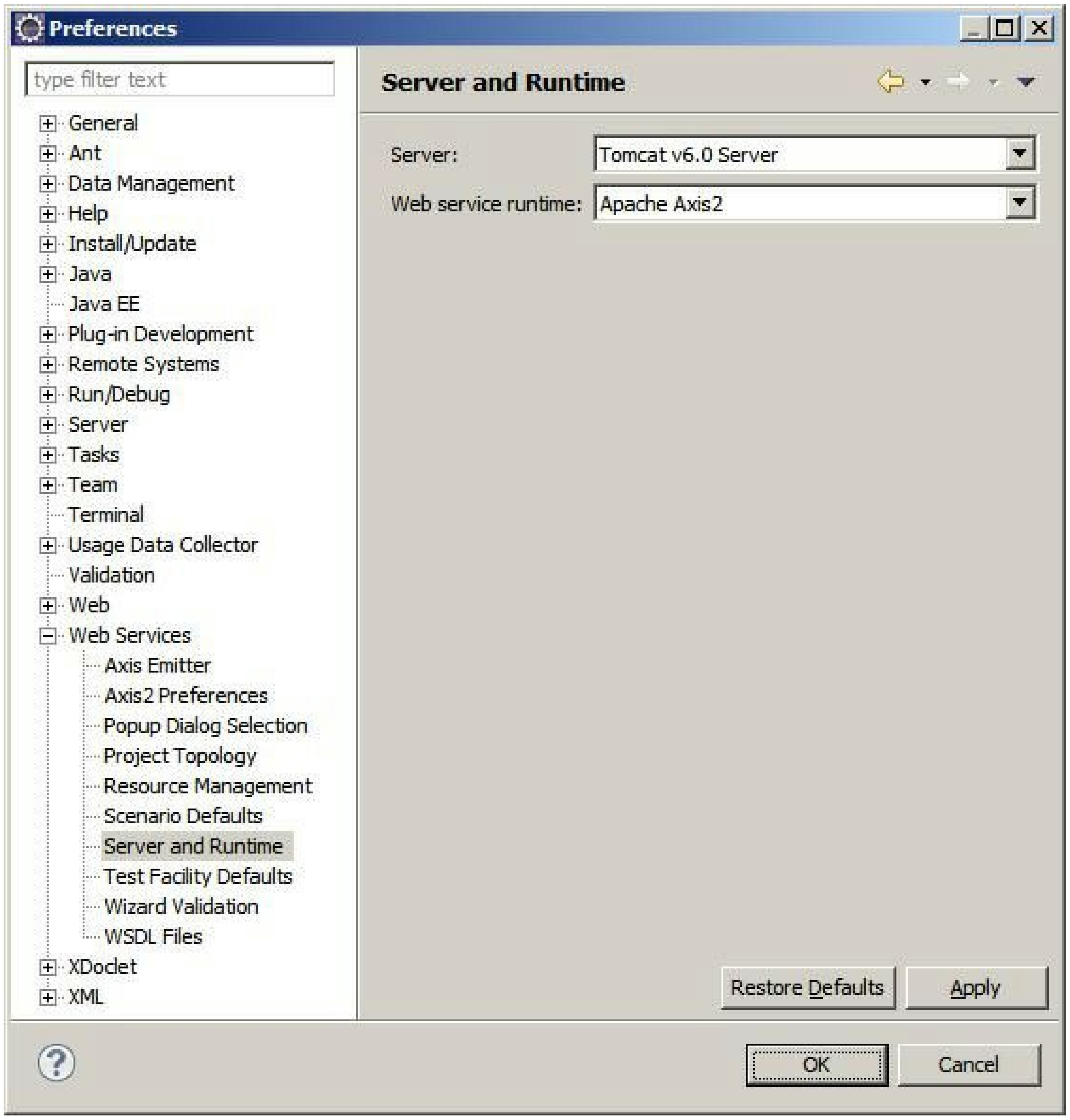
No client

Publish the Web service

Monitor the Web service

Select *Server and Runtime*. Choose the *Server as Tomcat v6.0 Server and Web service runtime as Apache Axis2*.

Figure 18. Axis2 preferences (Server and Runtime) in Eclipse



Server and Runtime

Server: Tomcat v6.0 Server

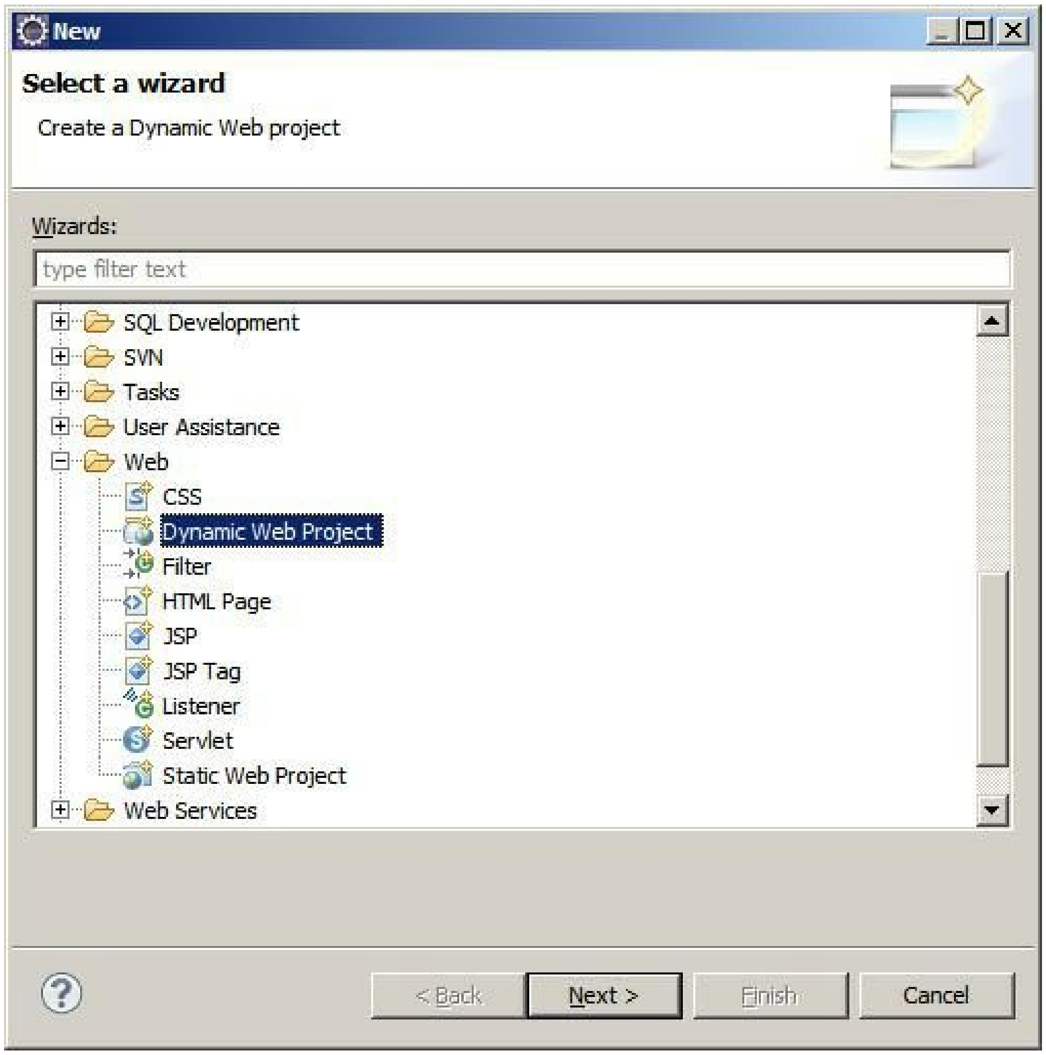
Web service runtime: Apache Axis2

Click *OK* to save the settings.

Creation of Dynamic Web project

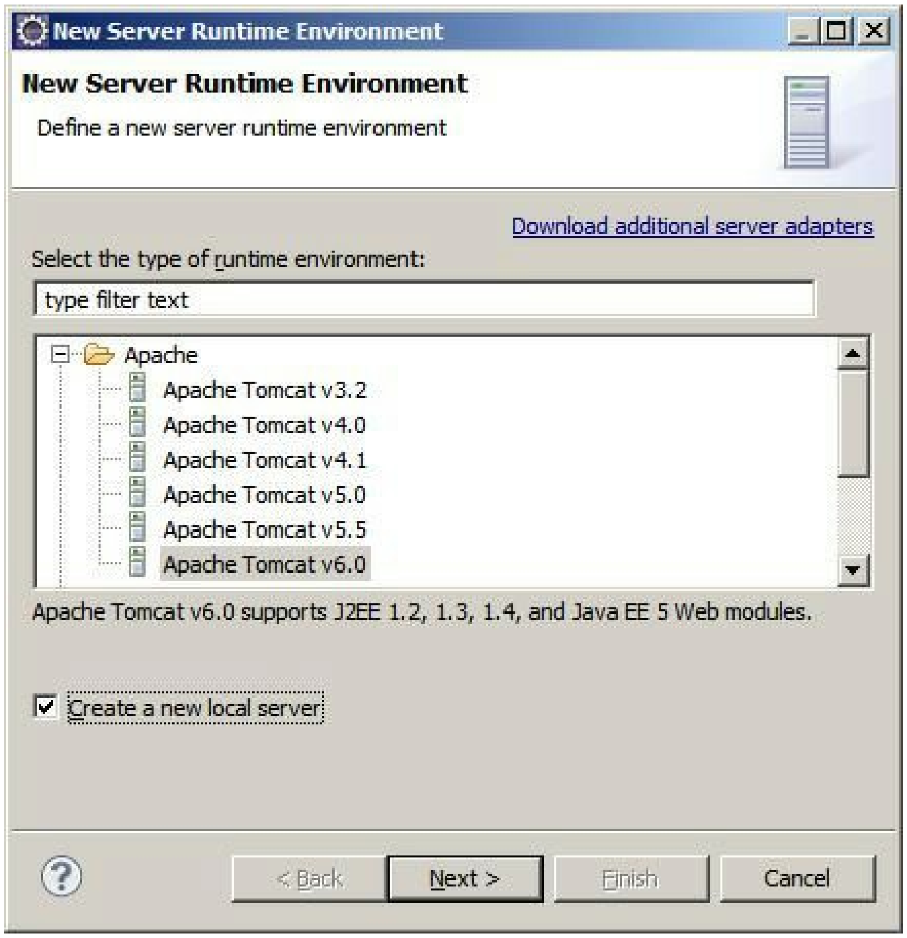
Create a new *Dynamic Web Project*.

Figure 19. Create new Dynamic Web project in Eclipse



Click *Next* and enter the project name as **StockQuoteWS**. Under *Target runtime* section, Click on *New*.

Figure 20. New Server Runtime in Eclipse


New Server Runtime Environment

Define a new server runtime environment

Select the type of runtime environment:

type filter text

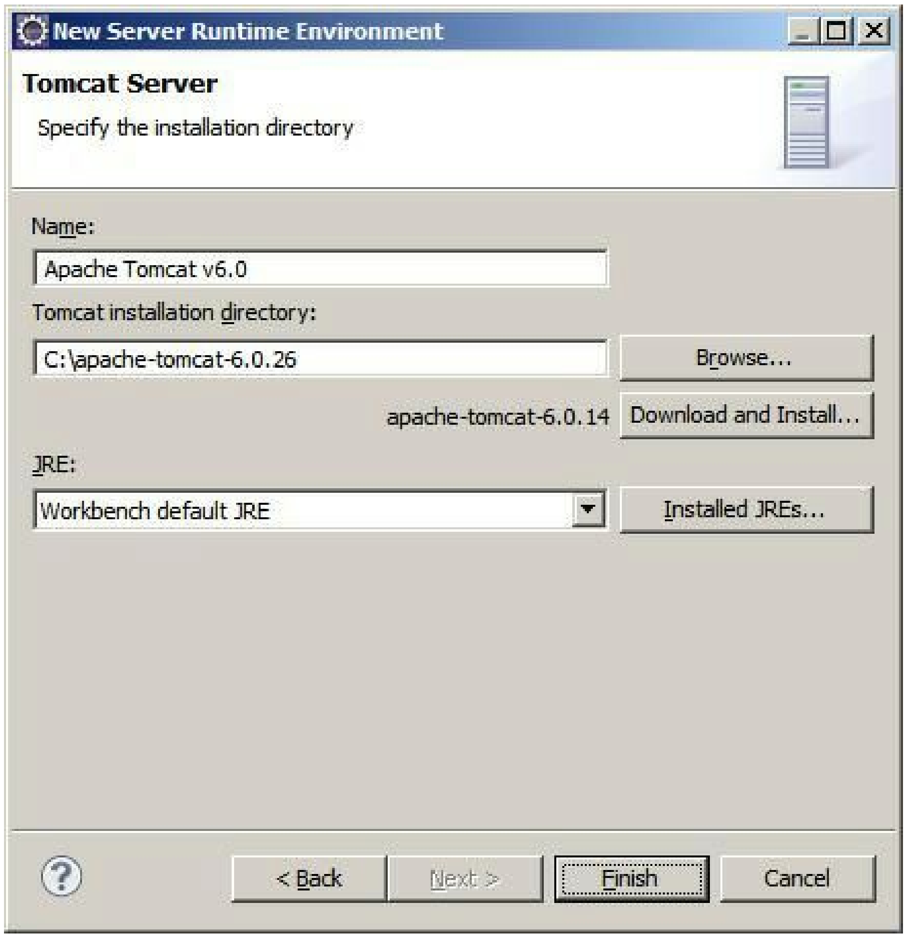
- Apache
  - Apache Tomcat v3.2
  - Apache Tomcat v4.0
  - Apache Tomcat v4.1
  - Apache Tomcat v5.0
  - Apache Tomcat v5.5
  - Apache Tomcat v6.0

Apache Tomcat v6.0 supports J2EE 1.2, 1.3, 1.4, and Java EE 5 Web modules.

[✓] Create a new local server

Select *Apache Tomcat v6.0* and select the *Create a new local server* check box and click *Next*. Under *Tomcat installation* directory, browse and select the tomcat installation directory (in our case, it is *C:\apache-tomcat-6.0.26*).

Figure 21. Tomcat Server preferences in Eclipse



New Server Runtime Environment

Tomcat Server

Specify the installation directory

Name: Apache Tomcat v6.0

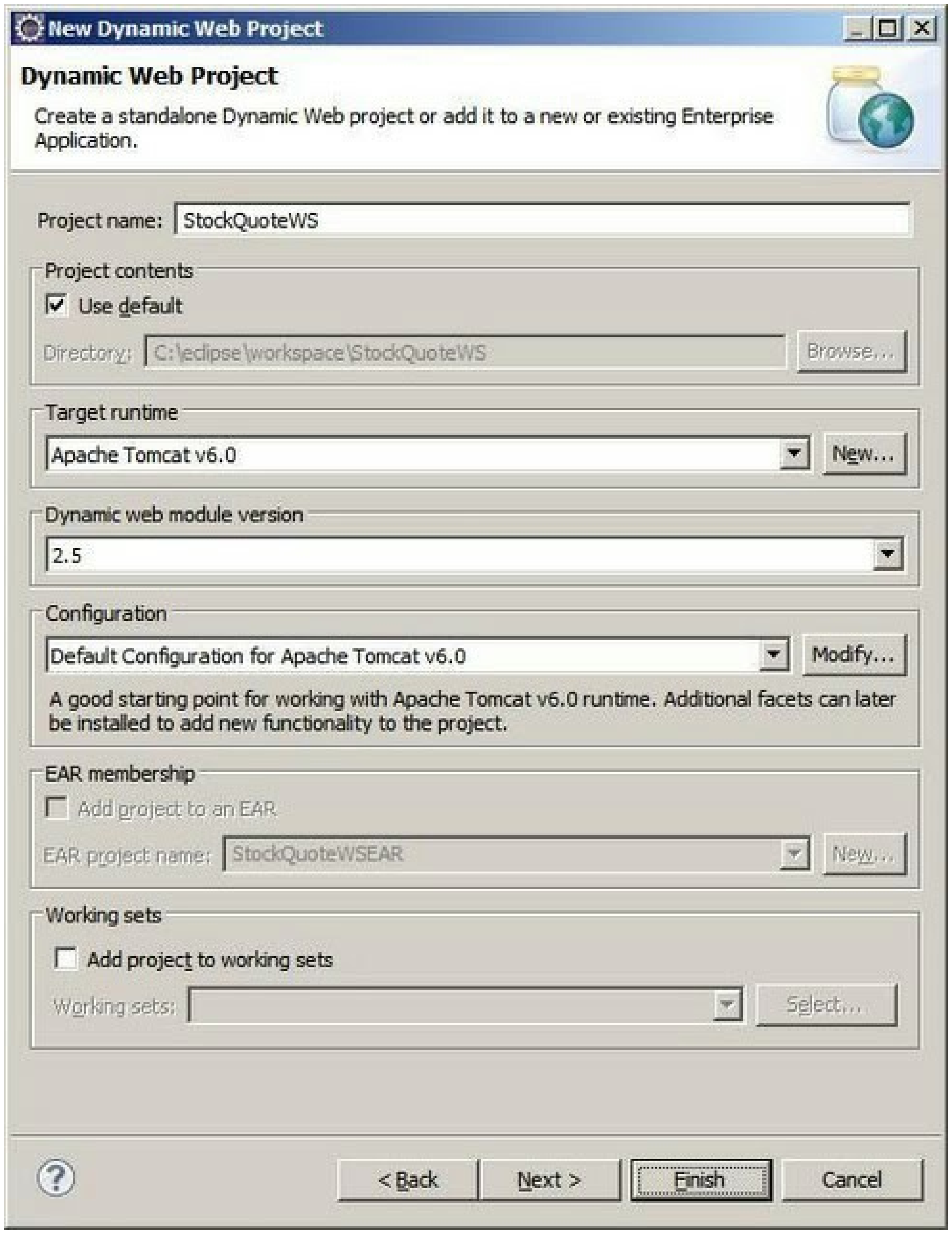
Tomcat installation directory: C:\apache-tomcat-6.0.26

apache-tomcat-6.0.14

JRE: Workbench default JRE

Click *Finish*. In the previous window, select *Apache Tomcat v6.0* under the *Target runtime* section.

Figure 22. Dynamic Web Project preferences in Eclipse


New Dynamic Web Project

Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: StockQuoteWS

Project contents

☑ Use default

Directory: C:\eclipse\workspace\StockQuoteWS

Target runtime: Apache Tomcat v6.0

Dynamic web module version: 2.5

Configuration: Default Configuration for Apache Tomcat v6.0

A good starting point for working with Apache Tomcat v6.0 runtime. Additional facets can later be installed to add new functionality to the project.

EAR membership

☐ Add project to an EAR

EAR project name: StockQuoteWSEAR

Working sets

☐ Add project to working sets

Working sets:

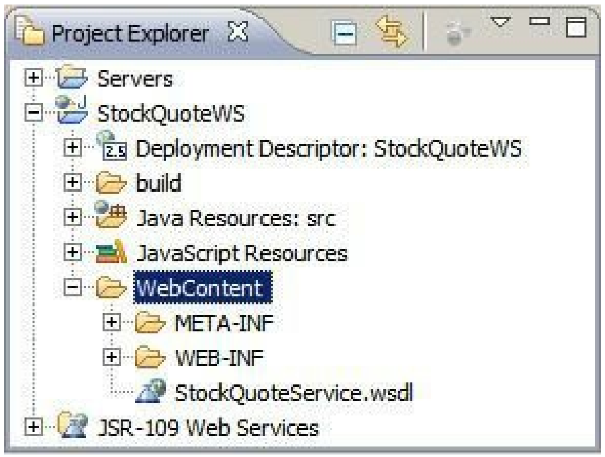
Click *Finish*. This will create the *StockQuoteWS* Dynamic Web Project. If you have not switched to *Web Perspective* in Eclipse after this, switch to Web Perspective (by selecting the menu *Window* > *Open Perspective* > *Web*). Copy the *StockQuoteService.wsdl* on to the *WebContent* folder. The *Project Explorer* will look like the one below.

Figure 23. Project Explorer of a Dynamic Web project in Eclipse

Generation of web service java skeleton

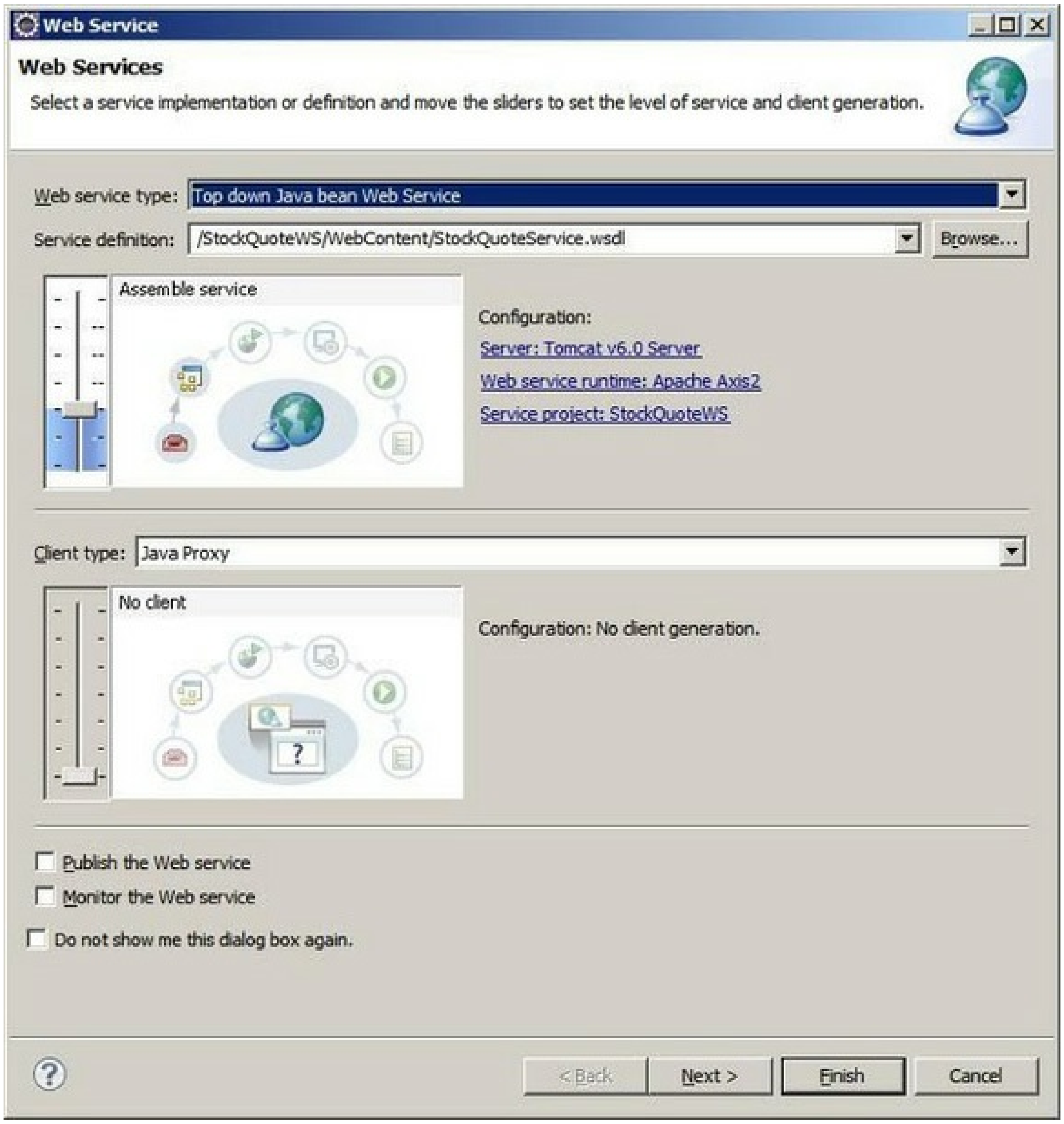
Right click on *StockQuoteService.wsdl* file and select *Web Services* > *Generate Java bean skeleton*. In the new popup window, check the *Server is Tomcat v6.0 Server, Web service runtime is Apache Axis2 and Service project is StockQuoteWS*.

Figure 24. Web services settings in Eclipse



Project Explorer

- Servers
- StockQuoteWS
  - Deployment Descriptor: StockQuoteWS
  - build
  - Java Resources: src
  - JavaScript Resources
  - WebContent
    - META-INF
    - WEB-INF
    - StockQuoteService.wsdl
- JSR-109 Web Services



Web Service

Web Services

Select a service implementation or definition and move the sliders to set the level of service and client generation.

Web service type: Top down Java bean Web Service

Service definition: /StockQuoteWS/WebContent/StockQuoteService.wsdl

Assemble service

Configuration:
Server: Tomcat v6.0 Server
Web service runtime: Apache Axis2
Service project: StockQuoteWS

Client type: Java Proxy

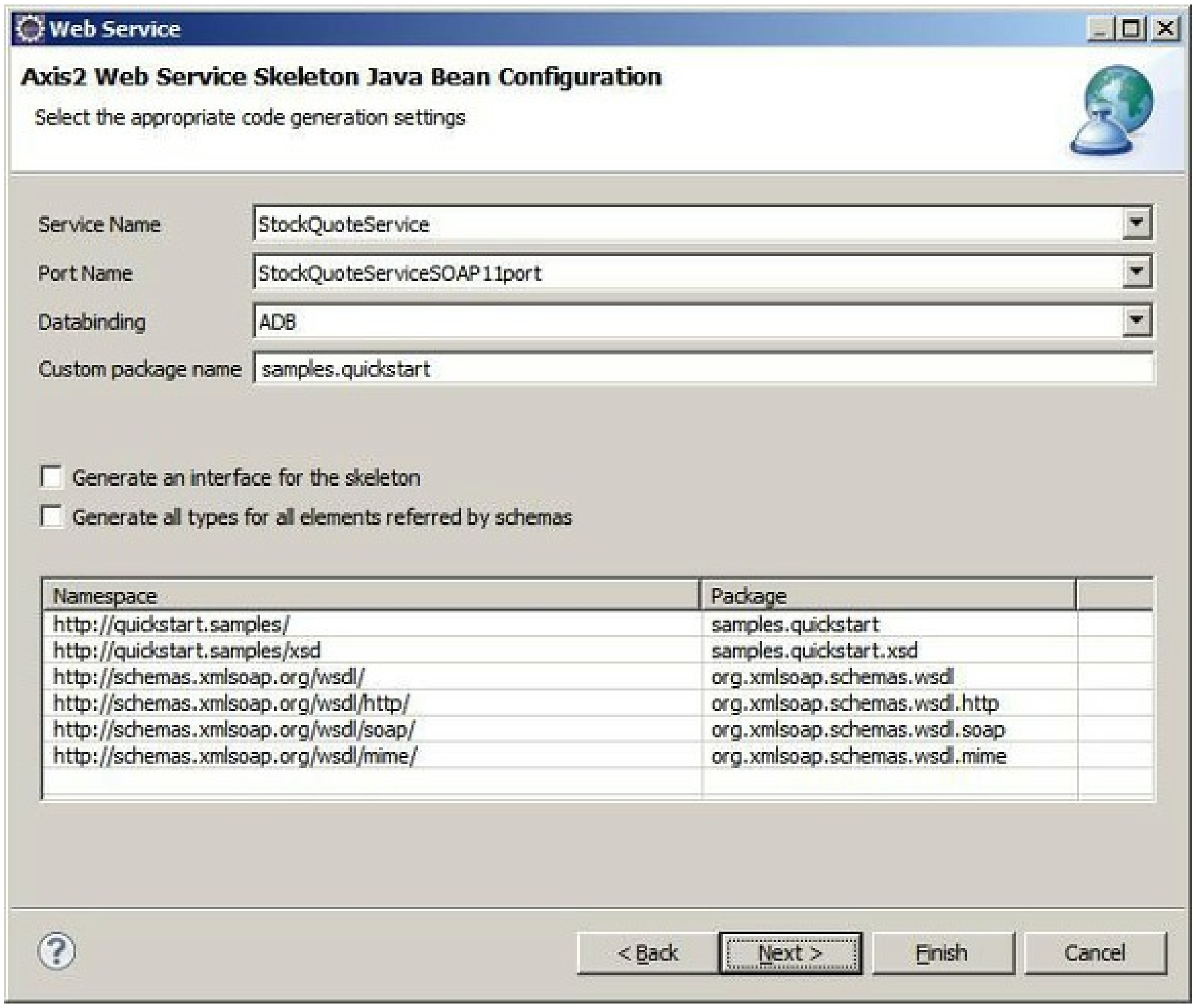
No client

Configuration: No client generation.

☐ Publish the Web service
☐ Monitor the Web service
☐ Do not show me this dialog box again.

Click *Next*. Here also we will use the *ADB* data binding.

Figure 25. Axis2 Web service Skeleton settings in Eclipse



Axis2 Web Service Skeleton Java Bean Configuration

Select the appropriate code generation settings

Service Name	StockQuoteService
Port Name	StockQuoteServiceSOAP11port
Databinding	ADB
Custom package name	samples.quickstart

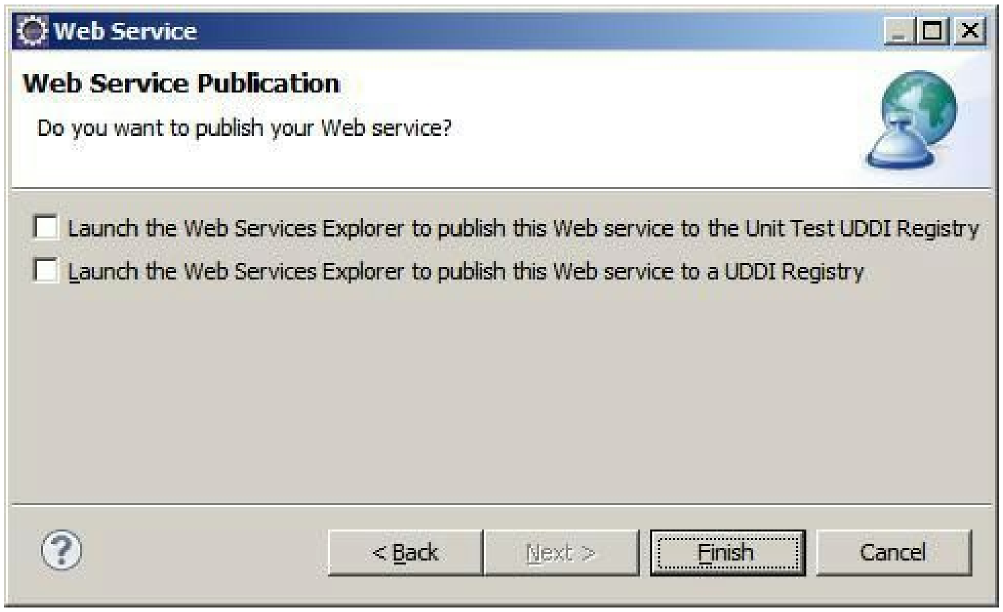
☐ Generate an interface for the skeleton

☐ Generate all types for all elements referred by schemas

Namespace	Package
http://quickstart.samples/	samples.quickstart
http://quickstart.samples/xsd	samples.quickstart.xsd
http://schemas.xmlsoap.org/wsdl/	org.xmlsoap.schemas.wsdl
http://schemas.xmlsoap.org/wsdl/http/	org.xmlsoap.schemas.wsdl.http
http://schemas.xmlsoap.org/wsdl/soap/	org.xmlsoap.schemas.wsdl.soap
http://schemas.xmlsoap.org/wsdl/mime/	org.xmlsoap.schemas.wsdl.mime

Click *Next* and then *Finish*.

Figure 26. Web service publication in Eclipse



Web Service Publication

Do you want to publish your Web service?

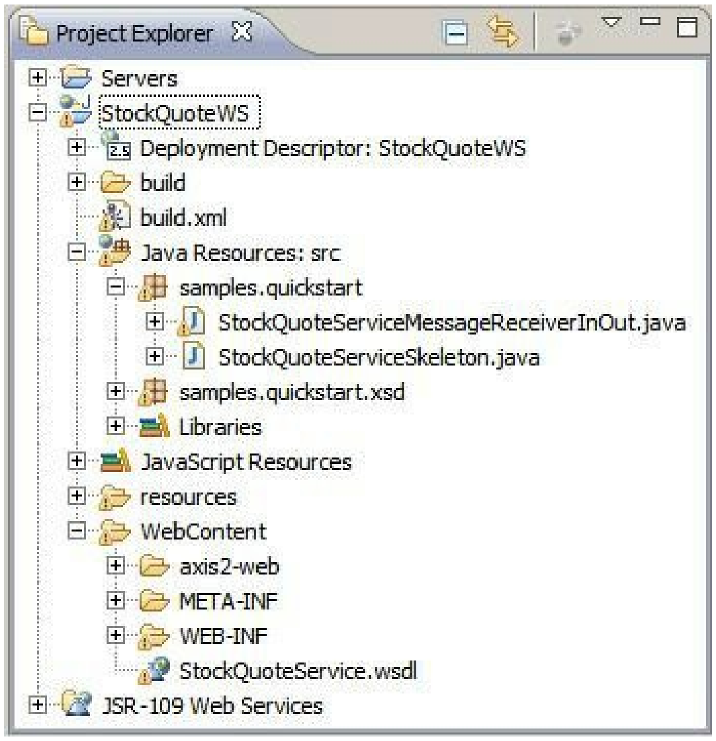
☐ Launch the Web Services Explorer to publish this Web service to the Unit Test UDDI Registry

☐ Launch the Web Services Explorer to publish this Web service to a UDDI Registry

The *Project Explorer* will look like the one in Figure 27.

Figure 27. Project Explorer in Eclipse

Completing the web service implementation



Hope you have noticed the new files being created. Here we are much concerned with two files; *viz.StockQuoteServiceSkeleton.java* and *build.xml*.

Open the class *samples.quickstart.StockQuoteServiceSkeleton* for editing and remove the line:

```
throw new
java.lang.UnsupportedOperationException("Please implement "
+ this.getClass().getName() + "#getPrice");
```

Add the below lines in place of the deleted line.

```
samples.quickstart.xsd.GetPriceResponse response = new
samples.quickstart.xsd.GetPriceResponse();
response.set_return(100.00d);
return response;
```

The whole class file should look like this:

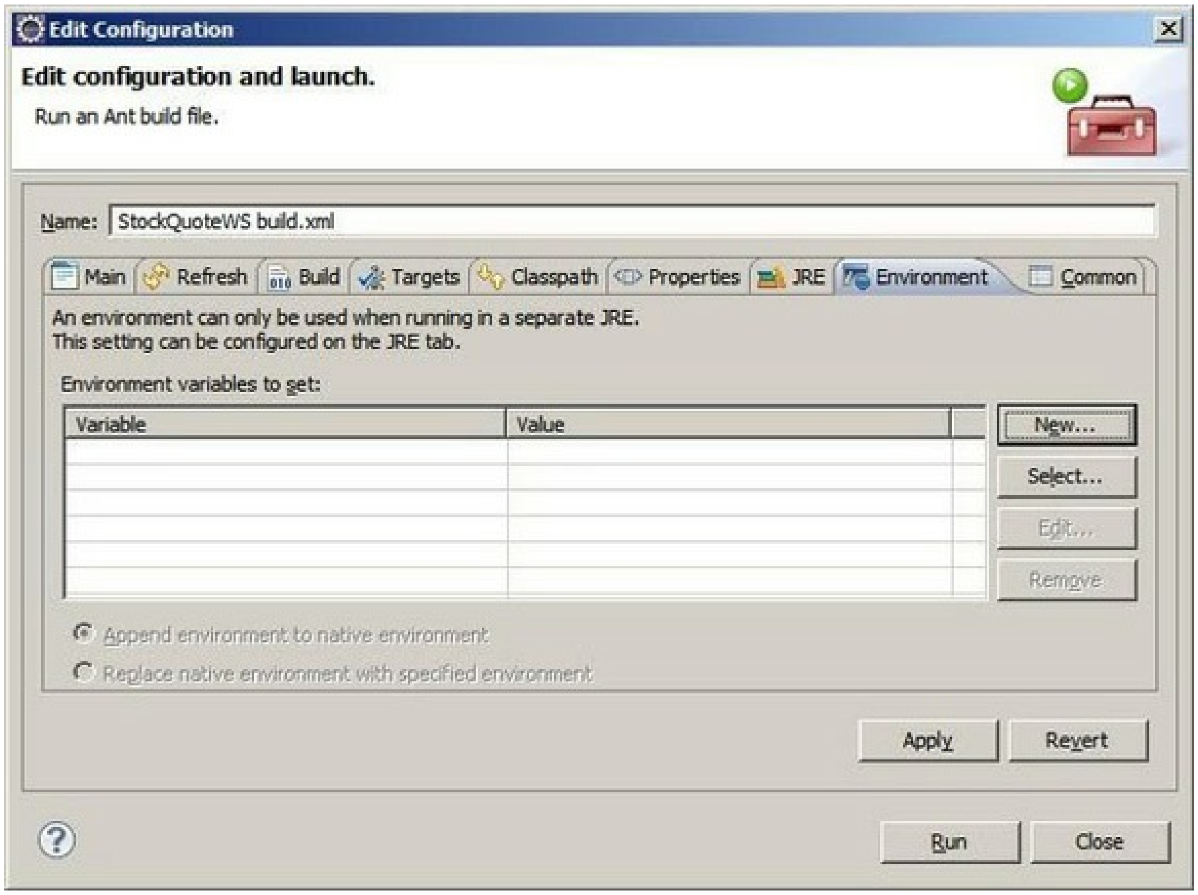
```
/**
* StockQuoteServiceSkeleton.java
*
* This file was auto-generated from WSDL
* by the Apache Axis2 version: 1.5.1 Built on : Oct 19, 2009 (10:59:00 EDT)
*/
package samples.quickstart;
/**
* StockQuoteServiceSkeleton java skeleton for the axisService
*/
public class StockQuoteServiceSkeleton implements
 StockQuoteServiceSkeletonInterface {
/**
* Auto generated method signature
*
* @param getPrice0
*/
    public samples.quickstart.xsd.GetPriceResponse getPrice
 (samples.quickstart.xsd.GetPrice getPrice0) {
 samples.quickstart.xsd.GetPriceResponse response = new
 samples.quickstart.xsd.GetPriceResponse();
        response.set_return(100.00d);
        return response;
    }
}
```

Creation of web service archive

So our service implementation is complete. For keeping the implementation simple, we will just return a value of *100.00* irrespective of the input *symbol* (*symbol* is the input of the operation *getPrice*, refer to the WSDL). You can provide a different implementation but should return an object of type *samples.quickstart.xsd.GetPriceResponse*.

Right-click on *build.xml* and select *Run As > Ant Build...* Select the *Environment* tab and then *New...* under *Environment variables to set* section.

Figure 28. ANT preferences in Eclipse



Edit Configuration

Edit configuration and launch.

Run an Ant build file.

Name: StockQuoteWS build.xml

An environment can only be used when running in a separate JRE.
This setting can be configured on the JRE tab.

Environment variables to set:

Variable	Value

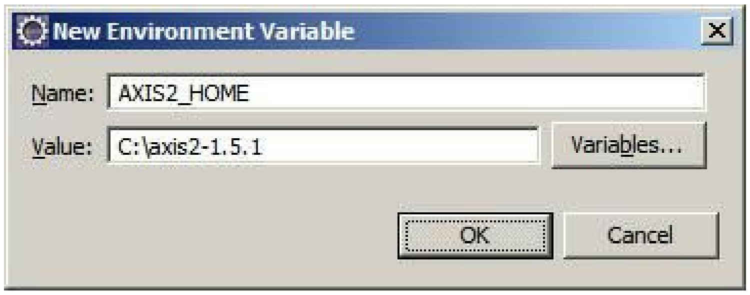
Append environment to native environment

Replace native environment with specified environment

In the new window, enter *AXIS2_HOME* in the *Name* field and *C:\axis2-1.5.1* in the *Value* field and click *OK*.

Figure 29. Axis2 environment value setting in Eclipse

Click *Apply* and then *Run* in the previous window.


New Environment Variable

Name: AXIS2_HOME

Value: C:\axis2-1.5.1

Configure Eclipse IDE to support Axis2 1.5.1

By default Eclipse IDE does not support Axis2 1.5.1 completely. So we need to make some changes in the components generated by the Eclipse's Axis2 plugin. For that we need to do the following tasks:

1. Open the web.xml file (under *StockQuoteWS* > *WebContent* > *WEB-INF* folder of *StockQuoteWS* Dynamic Web Project) and replace the below lines>

```
<servlet>
   <display-name>Apache-Axis Admin Servlet Web Admin</display-name>
   <servlet-name>AxisAdminServlet</servlet-name>
   <servlet-class>org.apache.axis2.transport.http.AxisAdminServlet</servlet-class>
   <load-on-startup>100</load-on-startup>
</servlet>
```

With:

```
<servlet>
   <display-name>Apache-Axis Admin Servlet Web Admin</display-name>
   <servlet-name>AxisAdminServlet</servlet-name>
   <servlet-class>org.apache.axis2.webapp.AxisAdminServlet</servlet-class>
   <load-on-startup>100</load-on-startup>
</servlet>
```

Note the change in the value of *servlet-class* attribute.

2. Extract *httpcomponents-core-4.0.1-bin.zip* (Apache HttpCore 4.0.1, mentioned in software required section under pre-requisites).
3. Copy *httpcore-4.0.1.jar* from the lib folder of the above extract and put it in *lib* folder (*lib* folder can be found under *StockQuoteWS* > *WebContent* > *WEB-INF* folder of StockQuoteWS Dynamic Web Project) of the StockQuoteWS Dynamic Web Project.

Hosting the web service archive

If the *Servers* view is not visible, open it using the menu shortcuts *Window* > *Show View* > *Other...* and select *Servers* from the *Server* section.

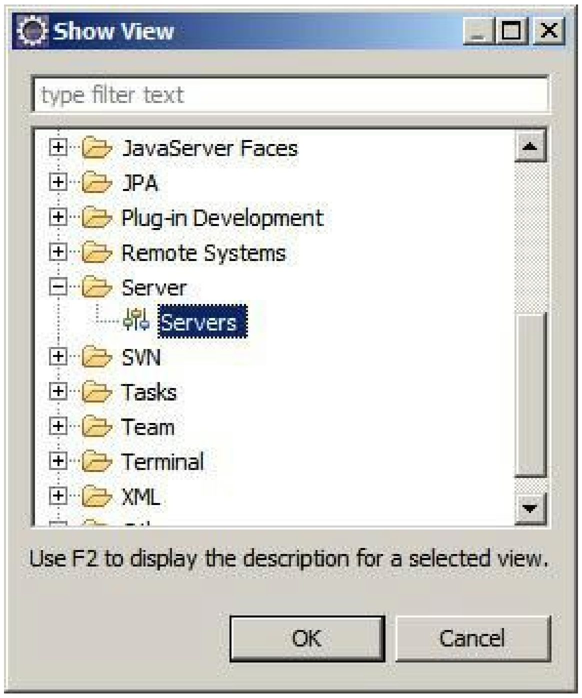
Figure 30. Views in Eclipse

This will open the Servers view.

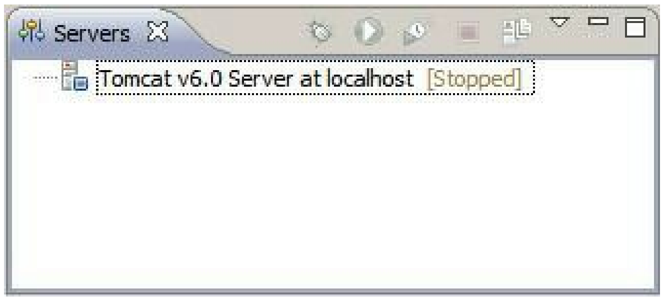
Figure 31. Servers view in Eclipse

Right click on the *Tomcat v6.0 Server at localhost* from the Servers view and select the *Add and Remove* option.

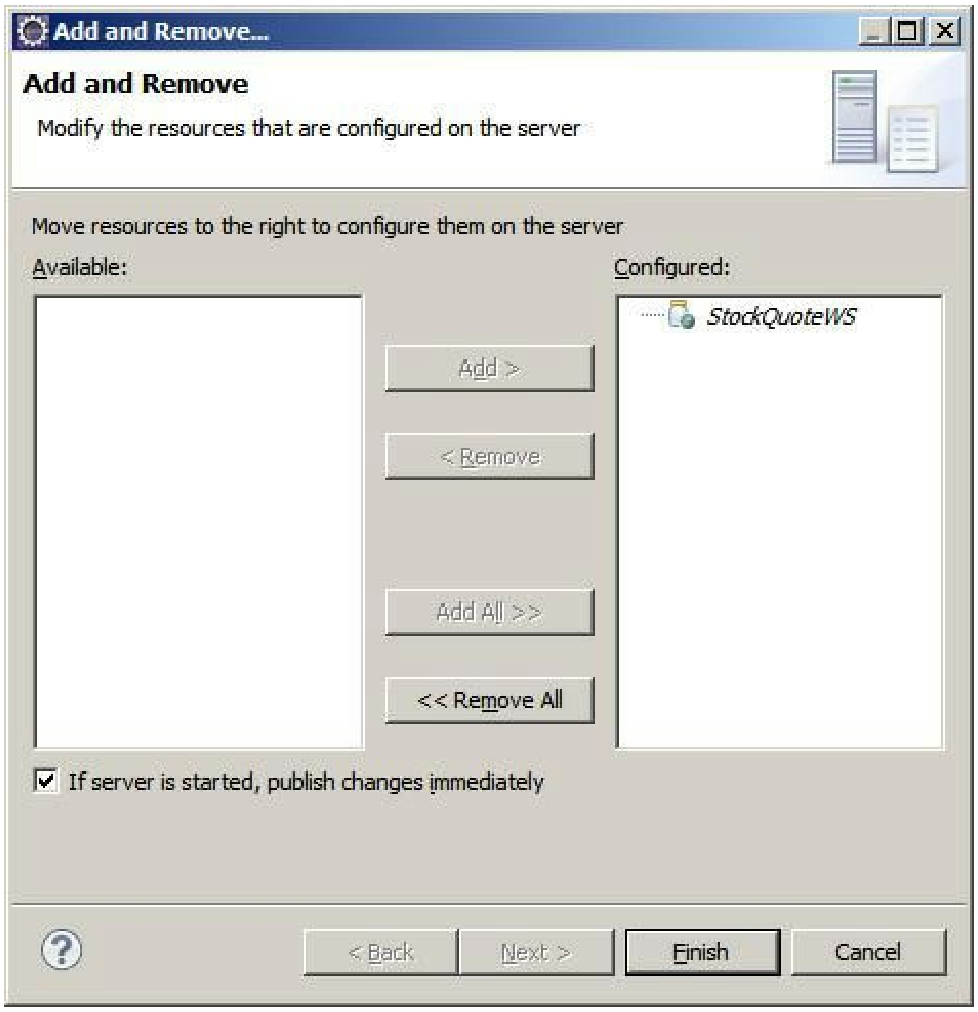
Figure 32. Adding projects to Servers in Eclipse



Use F2 to display the description for a selected view.



Tomcat v6.0 Server at localhost [Stopped]


Add and Remove...
Add and Remove
Modify the resources that are configured on the server
Move resources to the right to configure them on the server
Available:
Configured:
StockQuoteWS
Add >
< Remove
Add All >>
<< Remove All
If server is started, publish changes immediately
< Back
Next >
Finish
Cancel

Add the *StockQuoteWS* project from the *Available* section to the *Configured* section using the *Add* button. Click *Finish*.

Figure 33. Servers view with added projects in Eclipse

Again right-click on the *Tomcat v6.0 Server at localhost* from the *Servers* view and select the *Start* option. This will start the Tomcat server. Next, open a browser and open the address:
http://localhost:8080/StockQuoteWS/services/listServices.

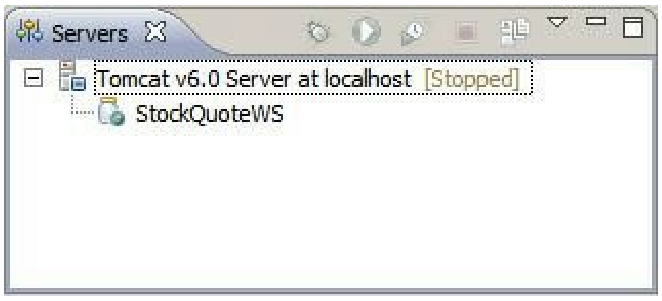
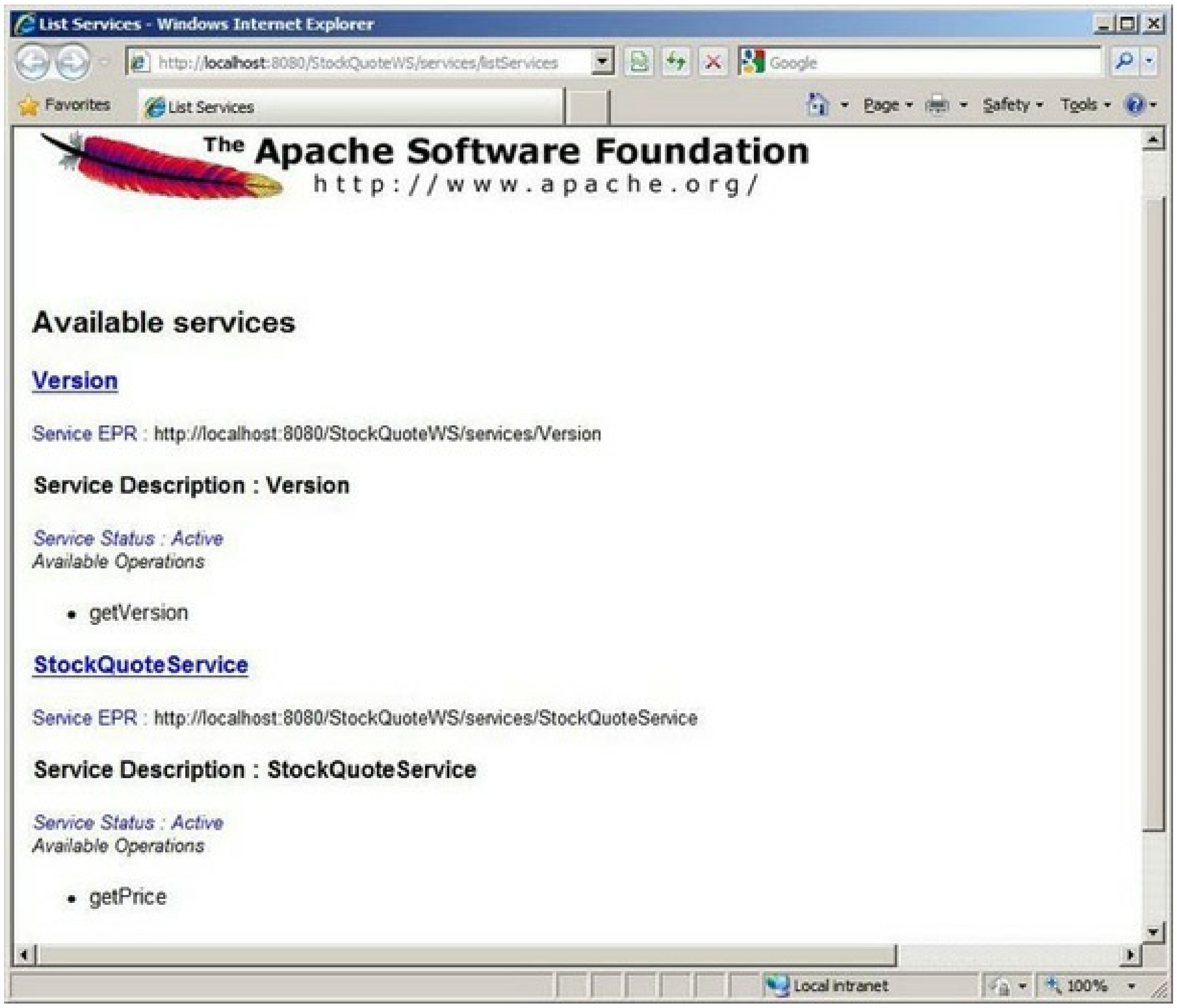


Figure 34. Web services deployed in Axis2


List Services - Windows Internet Explorer
http://localhost:8080/StockQuoteWS/services/listServices
Favorites
List Services
Page
Safety
Tools
The Apache Software Foundation
http://www.apache.org/
Local intranet
100%

Available services

Version

Service EPR : http://localhost:8080/StockQuoteWS/services/Version

Service Description : Version

Service Status : Active
Available Operations

- getVersion

StockQuoteService

Service EPR : http://localhost:8080/StockQuoteWS/services/StockQuoteService

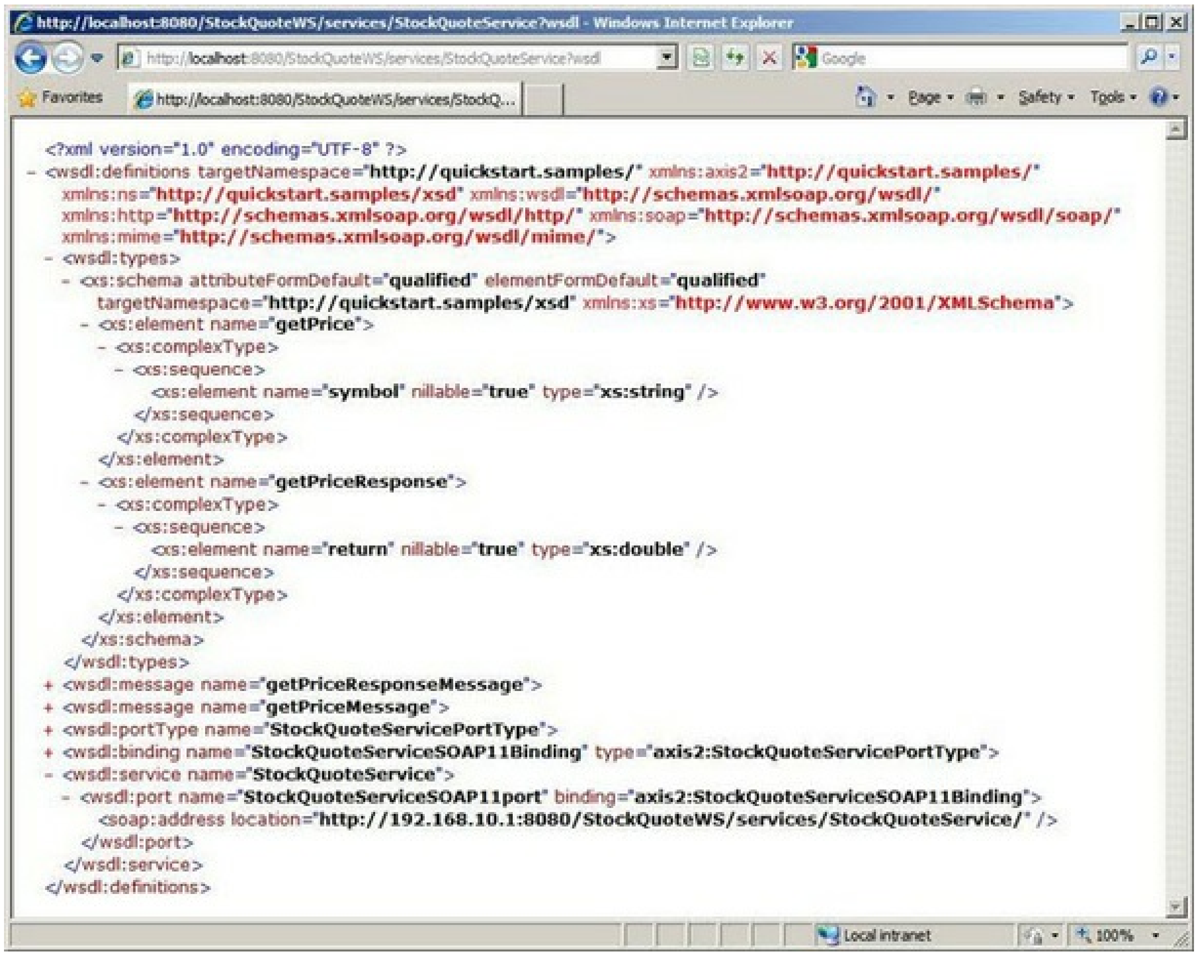
Service Description : StockQuoteService

Service Status : Active
Available Operations

- getPrice

This will list the web services which are deployed. Click on the *StockQuoteService* link to view the WSDL of the service.

Figure 35. WSDL file of a deployed server in Axis2


http://localhost:8080/StockQuoteWS/services/StockQuoteService?wsdl - Windows Internet Explorer
http://localhost:8080/StockQuoteWS/services/StockQuoteService?wsdl
Google
Favorites
http://localhost:8080/StockQuoteWS/services/StockQ...
Page
Safety
Tools
Local intranet
100%

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions targetNamespace="http://quickstart.samples/" xmlns:axis2="http://quickstart.samples/"
    xmlns:ns="http://quickstart.samples/xsd" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/">
  - <wsdl:types>
    - <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified"
        targetNamespace="http://quickstart.samples/xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema">
      - <xs:element name="getPrice">
        - <xs:complexType>
          - <xs:sequence>
              <xs:element name="symbol" nillable="true" type="xs:string" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      - <xs:element name="getPriceResponse">
        - <xs:complexType>
          - <xs:sequence>
              <xs:element name="return" nillable="true" type="xs:double" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:schema>
    </wsdl:types>
  + <wsdl:message name="getPriceResponseMessage">
  + <wsdl:message name="getPriceMessage">
  + <wsdl:portType name="StockQuoteServicePortType">
  + <wsdl:binding name="StockQuoteServiceSOAP11Binding" type="axis2:StockQuoteServicePortType">
  - <wsdl:service name="StockQuoteService">
    - <wsdl:port name="StockQuoteServiceSOAP11port" binding="axis2:StockQuoteServiceSOAP11Binding">
        <soap:address location="http://192.168.10.1:8080/StockQuoteWS/services/StockQuoteService/" />
      </wsdl:port>
    </wsdl:service>
  </wsdl:definitions>
```

The web service is ready to for service. Web service invocation will be the same as we explained earlier except for the change in the URL of the web service. The new web service URL will be: http://localhost:8080/StockQuoteWS/services/StockQuoteService.

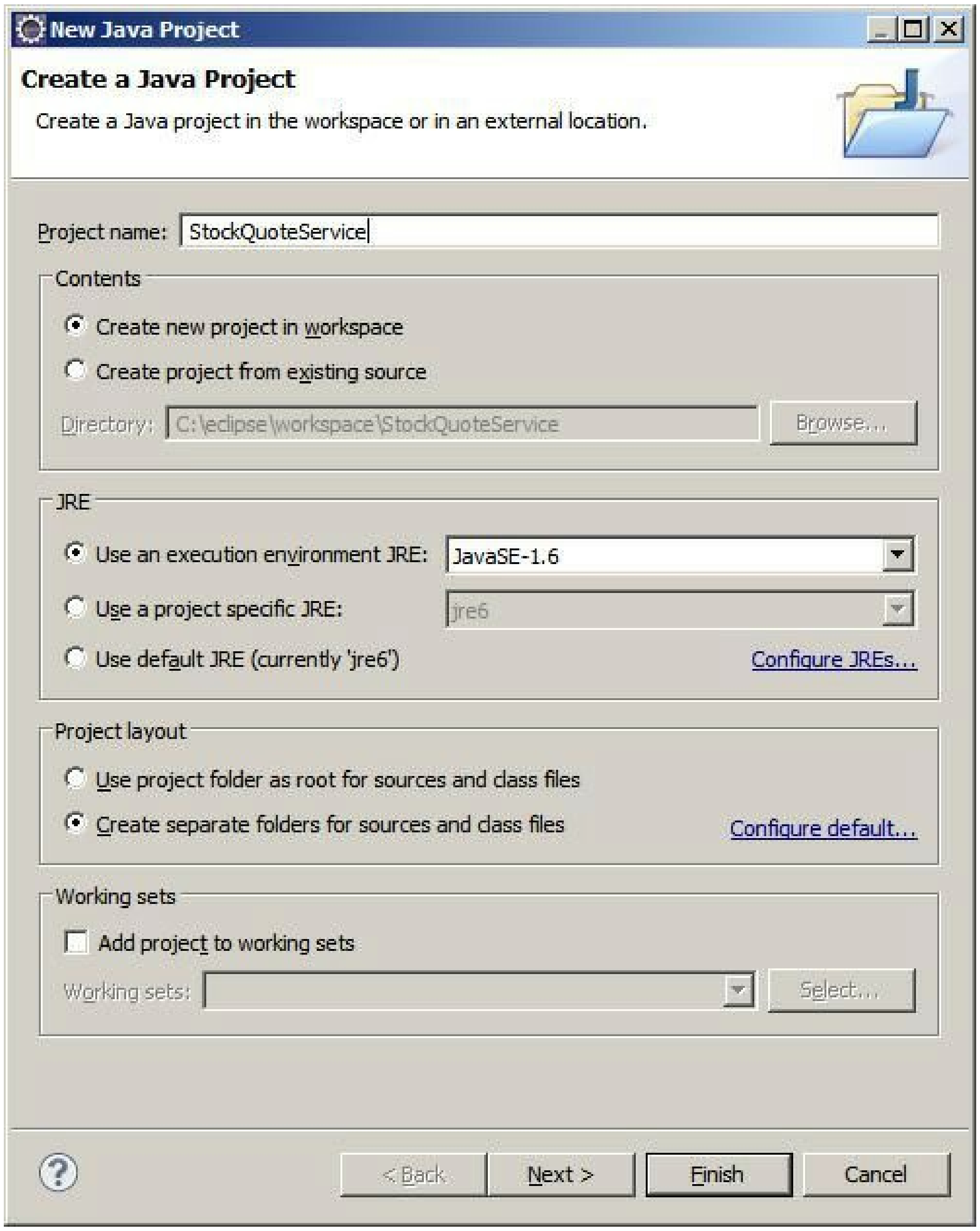
Creating WSDL file from a java class

In the previous sections, we started with an already available WSDL file (*StockQuoteService.wsdl*). How about creating our own WSDL file? Axis2 provides a tool to create a WSDL file from an existing java class file. Let us see how we can create our own WSDL file. This tool is "*Java2WSDL*" which can be found in the *bin* directory of Axis2 binary distribution.

The Axis2 *Java2WSDL tool* can be used to generate a WSDL file from a Java class. This tool will inspect the java class and creates operations based on the methods in the java class and the equivalent data types.

1. Create a new Java project with name **StockQuoteService** in Eclipse. It is not necessary that we should create a Java project in Eclipse. (Note: Our aim is to create a java class; you are free to follow any other way of creating a java class).

Figure 36. New Java project creation in Eclipse


New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: StockQuoteService

Contents

(●) Create new project in workspace

( ) Create project from existing source

Directory: C:\eclipse\workspace\StockQuoteService

JRE

(●) Use an execution environment JRE: JavaSE-1.6

( ) Use a project specific JRE: jre6

( ) Use default JRE (currently 'jre6')

Project layout

( ) Use project folder as root for sources and class files

(●) Create separate folders for sources and class files

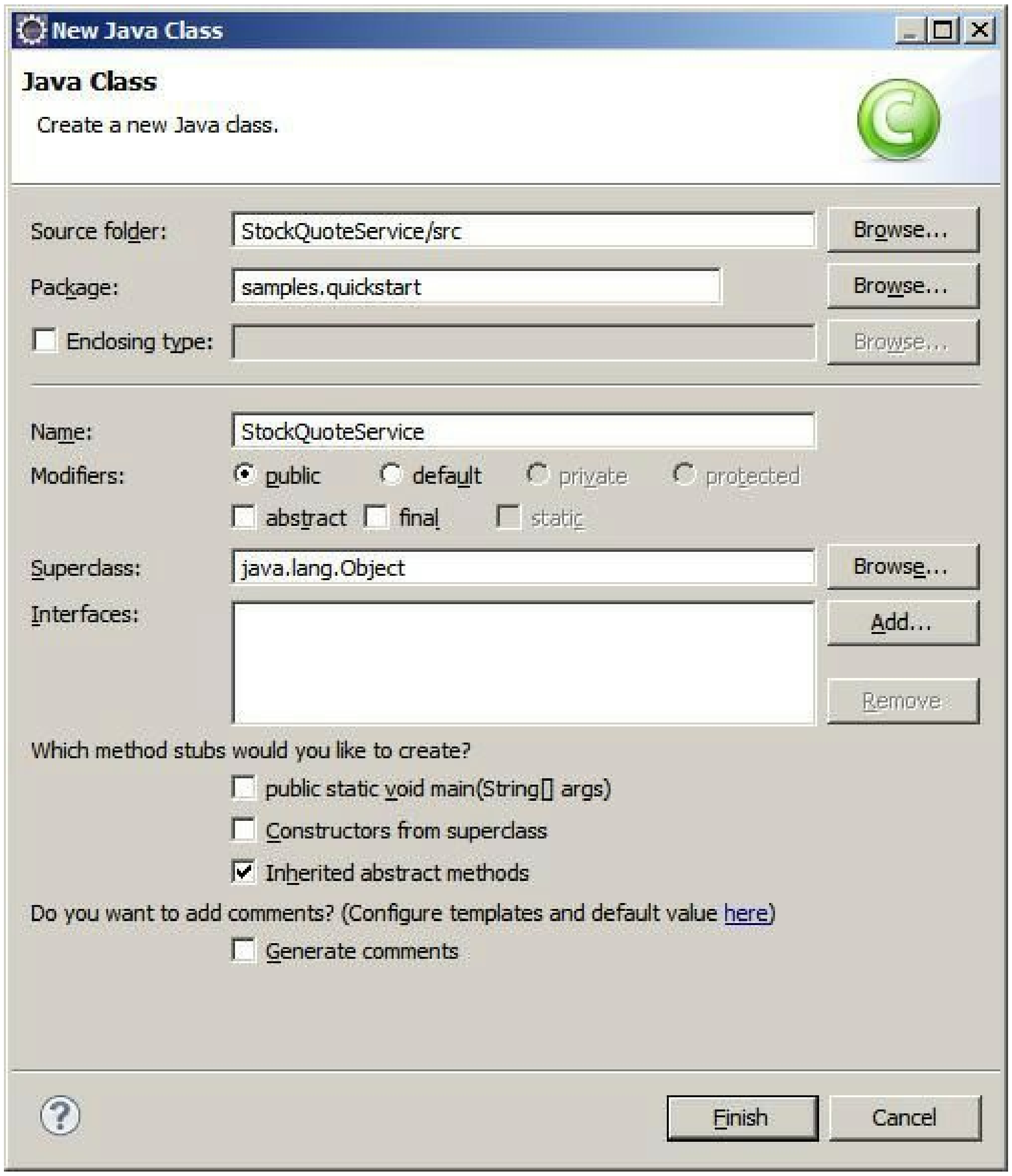
Working sets

[ ] Add project to working sets

Working sets:

2. Create a new java class *StockQuoteService* in package *samples.quickstart*

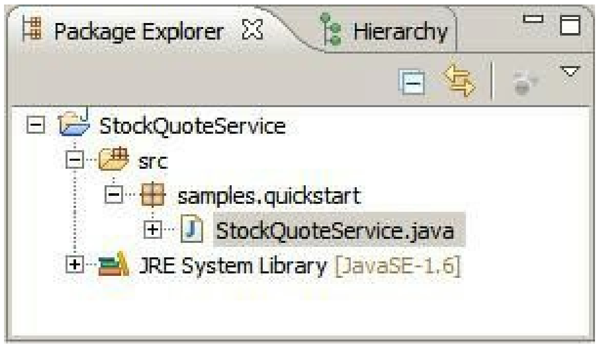
Figure 37. New Java class creation in Eclipse



Source folder: StockQuoteService/src
Package: samples.quickstart
☐ Enclosing type:
Name: StockQuoteService
Modifiers: ● public ○ default ○ private ○ protected
☐ abstract ☐ final ☐ static
Superclass: java.lang.Object
Interfaces:
Which method stubs would you like to create?
☐ public static void main(String[] args)
☐ Constructors from superclass
☑ Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
☐ Generate comments

3. Our new Java project should look like Figure 38.

Figure 38. Package explorer in Eclipse



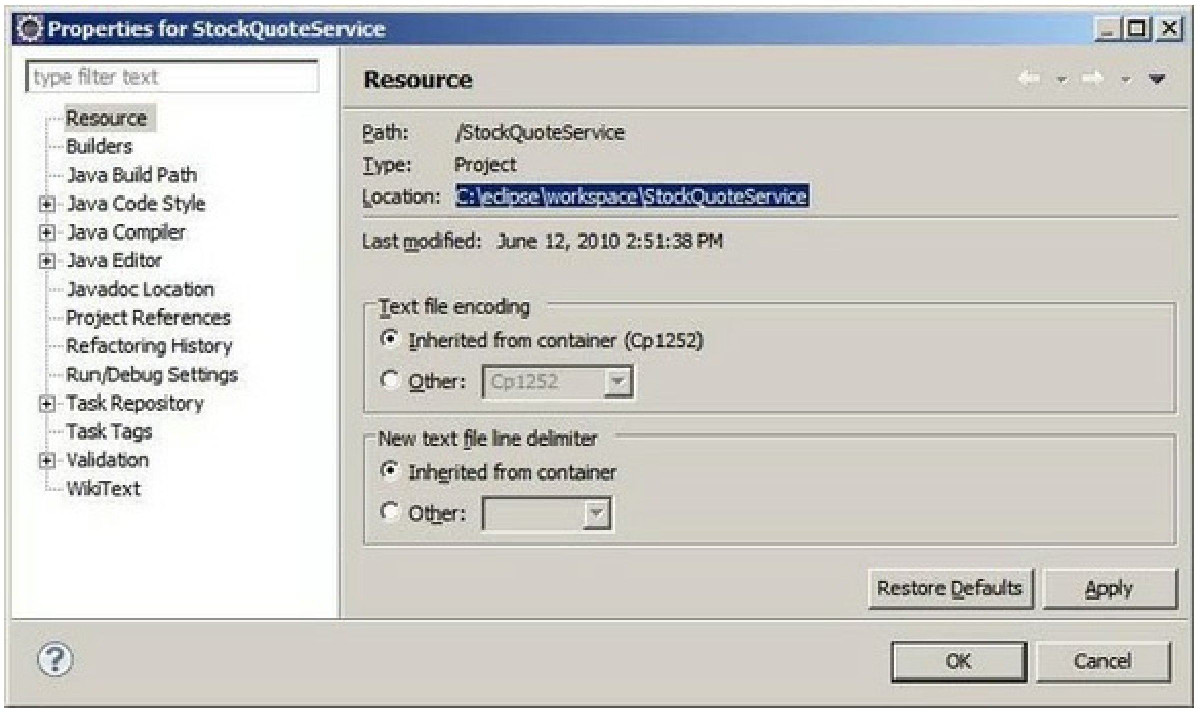
Create a new method *getPrice* in the *StockQuoteService* class. There is no need of providing the complete implementation logic. Just the correct return type and method parameters should be there and the class should not give any compile time errors. So our complete *StockQuoteService* class will look like the one below.

```
package samples.quickstart;
public class StockQuoteService {
    public double getPrice(String symbol) {
        return 0d;
    }
}
```

Here we did not provide the complete implementation, but to avoid compile time errors, we just returned a value.

4. We have to find out the bin folder of this Java project. This can be found by following the menu options *Project* > *Properties* > *Resource* and then the value of *Location*. (Make sure that you select the project name (*StockQuoteService*) in the *Project Explorer* or *Package Explorer* view in Eclipse).

Figure 39. Project properties in Eclipse


Properties for StockQuoteService

Resource

Path: /StockQuoteService
Type: Project
Location: C:\eclipse\workspace\StockQuoteService

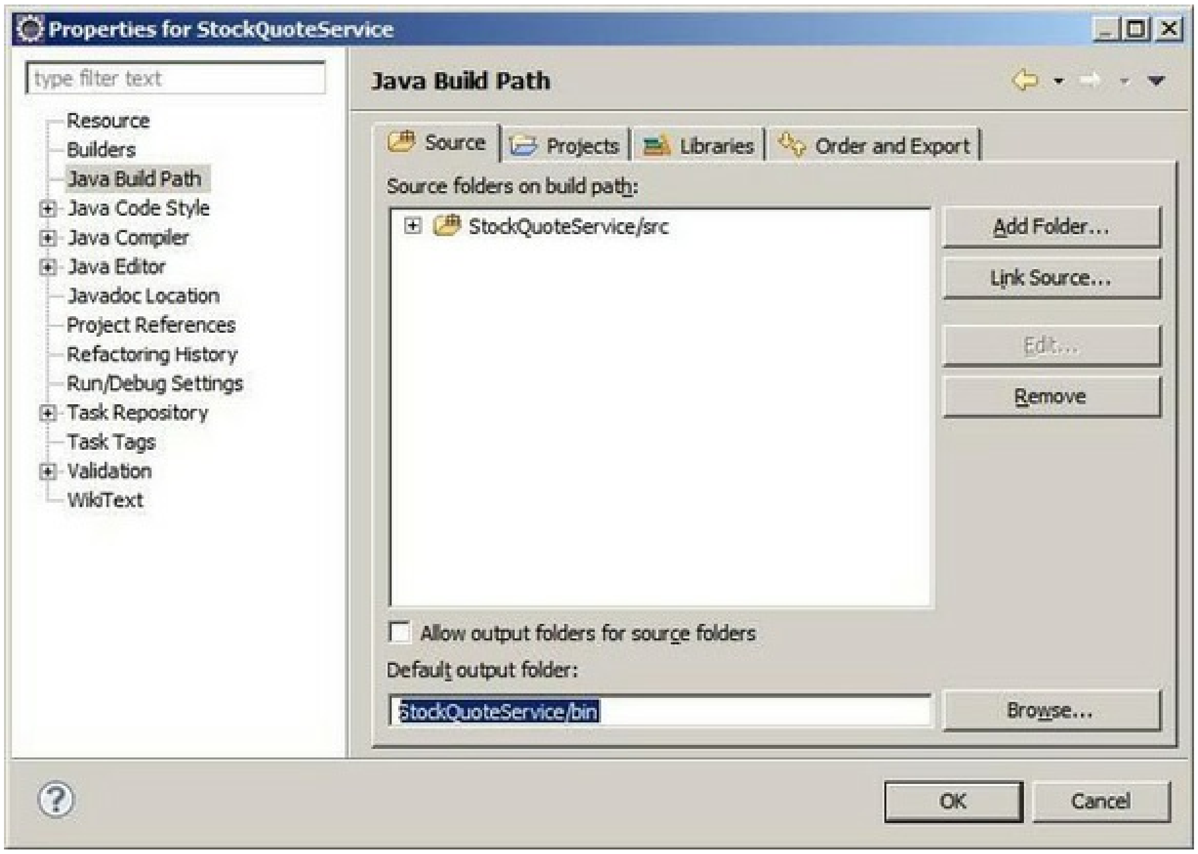
Last modified: June 12, 2010 2:51:38 PM

Text file encoding
(●) Inherited from container (Cp1252)
( ) Other: Cp1252

New text file line delimiter
(●) Inherited from container
( ) Other:

Note down the value of *Location*. In the same properties window, click on *Java Build Path* and then the *Source* tab. Check the value of *Default output folder*. The default output folder is pointing to folder bin the project folder. So in our case the bin directory will be *C:\eclipse\workspace\StockQuoteService\bin*.

Figure 40. Project build path in Eclipse



Source folders on build path:

StockQuoteService/src

Allow output folders for source folders

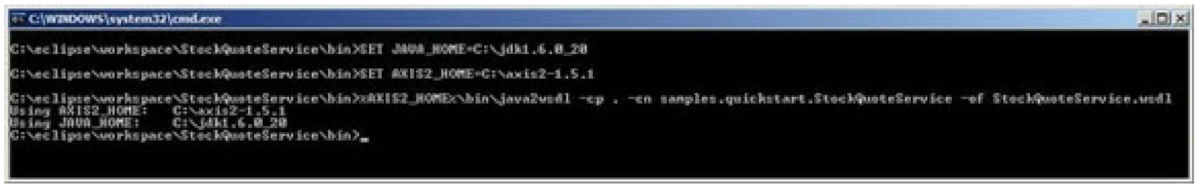
Default output folder:

StockQuoteService/bin

5. Open a command prompt window (using the shortcut *Start* > *Run* > *cmd*) and change the directory to the bin folder the above Java project. Set the *AXIS2_HOME and JAVA_HOME* environment variables. Run the following command:

```
%AXIS2_HOME%\bin\java2wsdl -cp . -cn
samples.quickstart.StockQuoteService -of
StockQuoteService.wsdl
```

Figure 41. WSDL creation from Java class



```
C:\eclipse\workspace\StockQuoteService\bin>SET JAVA_HOME=C:\jdk1.6.0_20

C:\eclipse\workspace\StockQuoteService\bin>SET AXIS2_HOME=C:\axis2-1.5.1

C:\eclipse\workspace\StockQuoteService\bin>%AXIS2_HOME%\bin\java2wsdl -cp . -cn samples.quickstart.StockQuoteService -of StockQuoteService.wsdl
Using AXIS2_HOME:   C:\axis2-1.5.1
Using JAVA_HOME:    C:\jdk1.6.0_20
C:\eclipse\workspace\StockQuoteService\bin>_
```

Where the options include those from Table 2 below.

Table 2. Java2WSDL options

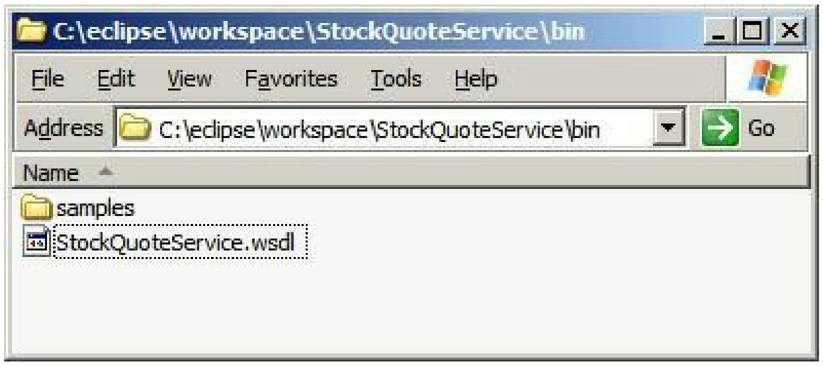
Option syntax	Description
`-cp <class path uri>`	list of classpath entries – (urls)
`-cn <fully qualified class name>`	fully qualified class name of the java class from which the WSDL file has to be generated

Option syntax	Description
`-of <output file name>`	output file name for the WSDL

There are other options also available with the *JAVA2WSDL* tool and can be seen by just invoking *%AXIS2_HOME%\bin\java2wsdl* command without any options. (Note: WSDL file can be generated from the Eclipse as well using Eclipse's inbuilt web service plugins.)

6. By the invocation of the above command, a WSDL file (*StockQuoteService.wsdl*) will be generated.

Figure 42. Generated components of JAVA2WSDL command


C:\eclipse\workspace\StockQuoteService\bin
File
Edit
View
Favorites
Tools
Help
Address
C:\eclipse\workspace\StockQuoteService\bin
Go
Name
samples
StockQuoteService.wsdl

This WSDL is the same as the WSDL file we have been using from the beginning.

Configuring TCP/IP Monitor

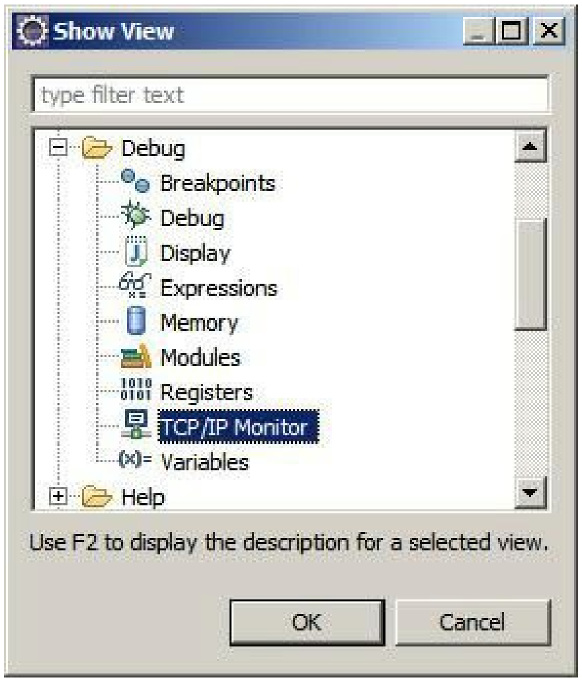
What if you want to see the actual SOAP request and response message? A TCP/IP monitor can be configured in eclipse to watch the request and response message of the web service. This can be achieved in the following steps below:

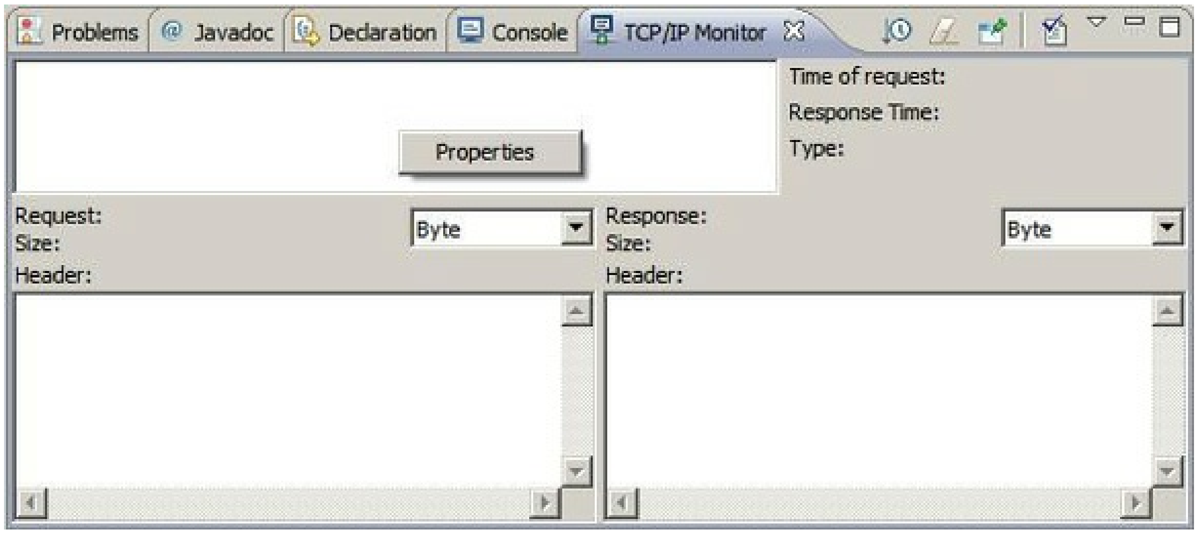
1. In Eclipse IDE, using the menu options *Window* > *Show View* > *Other*, select *TCP/IP Monitor* under the *Debug* category. This will open the TCP/IP Monitor view.

Figure 43. Views in Eclipse

2. Right click on the white area and select *Properties*.

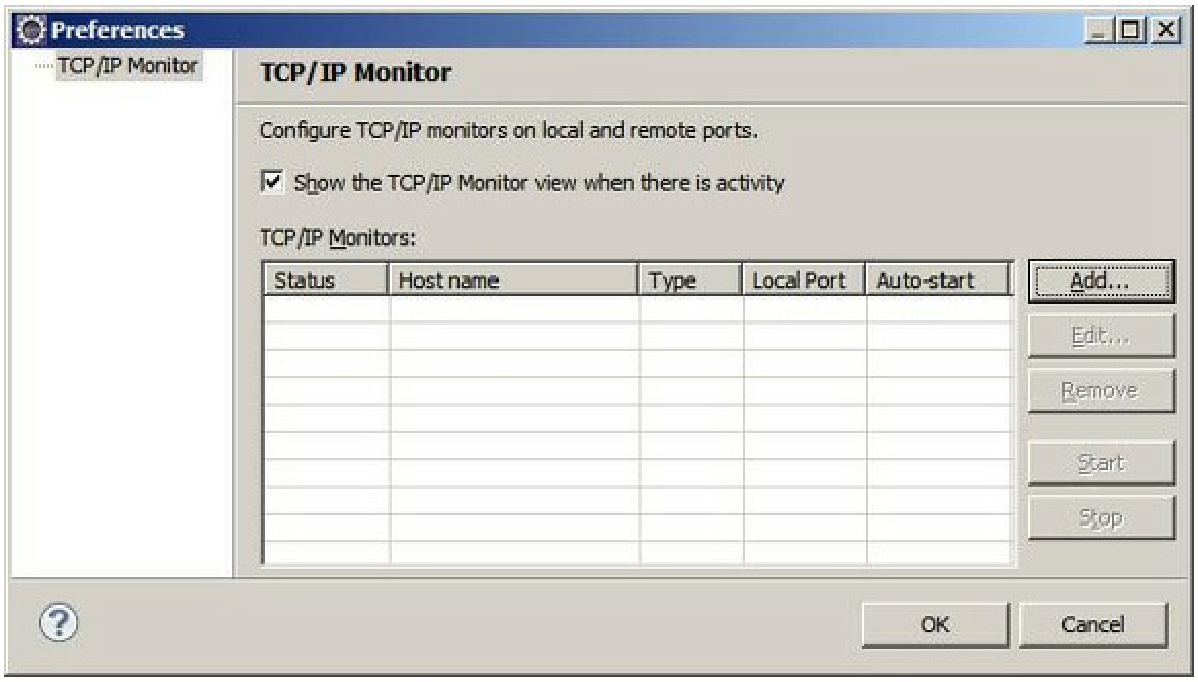
Figure 44. TCP/IP Monitor view in Eclipse





3. In the *Preferences* window, click *Add*.

Figure 45. TCP/IP Monitor preferences in Eclipse



Configure TCP/IP monitors on local and remote ports.

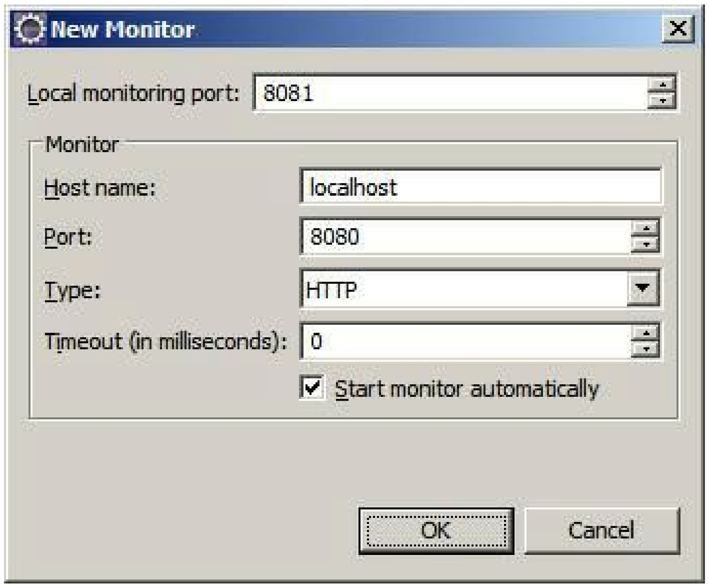
[✓] Show the TCP/IP Monitor view when there is activity

TCP/IP Monitors:

Status	Host name	Type	Local Port	Auto-start

4. Enter *Local monitoring port* as *8081* and *Host name* as *localhost* and *Port 8080* and select *Start monitor automatically*. By doing this we are actually creating a proxy for the server running on localhost:8080. All the requests and responses to and from localhost:8080 will travel through localhost:8081. This allows us to view the messages.

Figure 46. New TCP/IP Monitor creation in Eclipse



Local monitoring port: 8081
Host name: localhost
Port: 8080
Type: HTTP
Timeout (in milliseconds): 0
[✓] Start monitor automatically

5. Go back to our client code (*samples.quickstart.client.StockQuoteClient*) and change

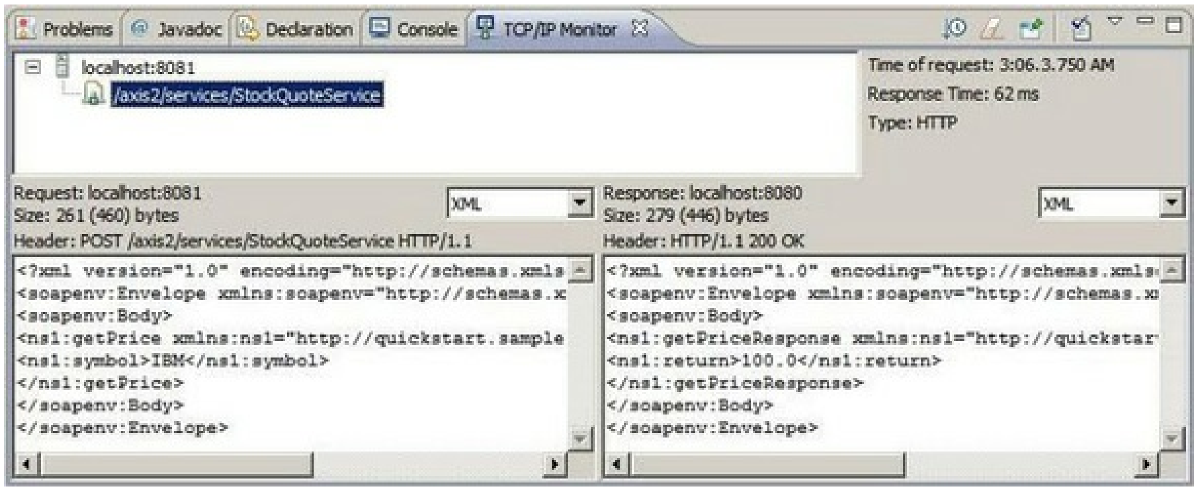
```
samples.quickstart.StockQuoteServiceStub
stub = new
samples.quickstart.StockQuoteServiceStub();
```

To:

```
samples.quickstart.StockQuoteServiceStub stub = new
samples.quickstart.StockQuoteServiceStub("http://localhost:8081/axis2
/services/StockQuoteService");
```

6. Change the view type to *XML* from *Byte* in TCP/IP Monitor window. Run the client program again and watch the TCP/IP Monitor. We can see the actual SOAP request and response messages.

Figure 47. TCP/IP Monitor request/response view in Eclipse



Time of request: 3:06.3.750 AM
Response Time: 62 ms
Type: HTTP

Request: localhost:8081
Size: 261 (460) bytes
Header: POST /axis2/services/StockQuoteService HTTP/1.1
XML

```
<?xml version="1.0" encoding="http://schemas.xmls
<soapenv:Envelope xmlns:soapenv="http://schemas.x
<soapenv:Body>
<ns1:getPrice xmlns:ns1="http://quickstart.sample
<ns1:symbol>IBM</ns1:symbol>
</ns1:getPrice>
</soapenv:Body>
</soapenv:Envelope>
```

Response: localhost:8080
Size: 279 (446) bytes
Header: HTTP/1.1 200 OK
XML

```
<?xml version="1.0" encoding="http://schemas.xmls
<soapenv:Envelope xmlns:soapenv="http://schemas.x
<soapenv:Body>
<ns1:getPriceResponse xmlns:ns1="http://quickstar
<ns1:return>100.0</ns1:return>
</ns1:getPriceResponse>
</soapenv:Body>
</soapenv:Envelope>
```

Back to top

Conclusion

We have seen how we can create a web service server components from an existing WSDL file, how to invoke it using Axis2. We have also seen how a WSDL file can be generated from an existing java class using Axis2. Hope you have understood the tasks. Please feel free to submit your queries and comments. We have seen how we can create a web service server components from an existing WSDL file, how to invoke it using Axis2. We have also seen how a WSDL file can be generated from an existing java class using Axis2. Hope you have understood the tasks. Please feel free to submit your queries and comments.