

Oracle® Agile Engineering Data Management

Creating an Agile e6 WebService in JDeveloper 11g

Andre Guldi

Agile PLM Product Management

Oracle Germany

September 2009

CONTENTS

Getting started	2
Step 1 Setting up JDeveloper.....	2
Exercise.....	2
Creating an Agile e6 WebService in JDeveloper 11g	3
Step 1: Creating a New Application and Project	3
Exercise.....	3
Step 2 : Add a Plain Old Java Object (POJO) to contain the Web Service Method	5
Exercise.....	5
Step 3: Test the Web Service.....	14
Exercise.....	14

Getting started

In this tutorial you will learn how to create your own custom Web Services for accessing and retrieving data inside Agile e6 using Oracle JDeveloper 11g and ADF (Application Development Framework) components.

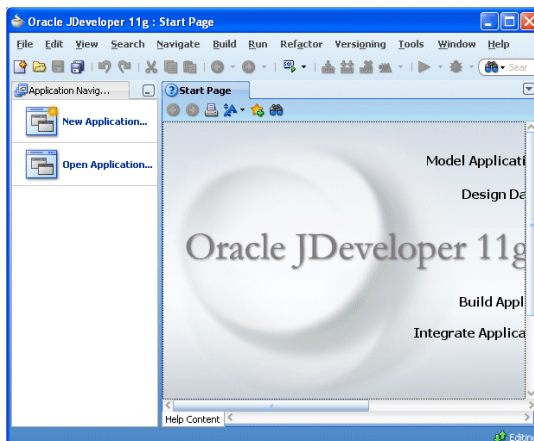
Note **Note that samples are provided for guidance only and should be tested in a development system before being applied on a production system. The sample code is not supported.**

Step 1 Setting up JDeveloper

Note The examples below are based on Version 11g of JDeveloper and Agile EDM Version e6.1. The code fragments and library names may differ for older versions of Agile EDM!

Exercise

1. Download Oracle JDeveloper 11g from the Oracle Technology Network (<http://www.oracle.com/technology/index.html>)
2. Install Oracle JDeveloper 11g as described in the Installation Manual on Oracle Technology Network - http://download.oracle.com/docs/cd/E12839_01/install.11111/e13666/toc.htm
3. After you have installed Oracle JDeveloper 11g on your computer, select Start > All Programs > Oracle WebLogic > JDeveloper Studio 11.1.1.0.0 to open JDeveloper
4. If the Migrate User Settings dialog box opens, click NO. If prompted for a User Role, choose Default and click OK.
5. Close the Tip of the Day window. The JDeveloper IDE should now be displayed.



Creating an Agile e6 WebService in JDeveloper 11g

Note The examples below are based on Version 11g of JDeveloper and Agile EDM Version e6.1. The code fragments and library names may differ for older versions of Agile EDM!

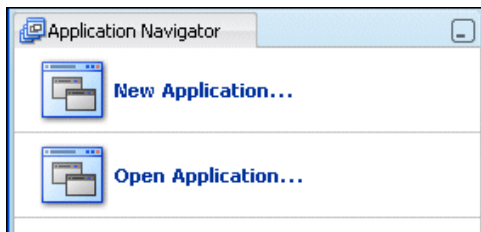
This chapter explains how to create an Agile e6 WebService in JDeveloper 11g:

1. **Step: Creating a new application and project**
 - Exercise
2. **Step: Add a Plain Old Java Object (POJO) to contain the Web Service Method**
 - Exercise
3. **Step: Test the Web Service**
 - Exercise

Step 1: Creating a New Application and Project

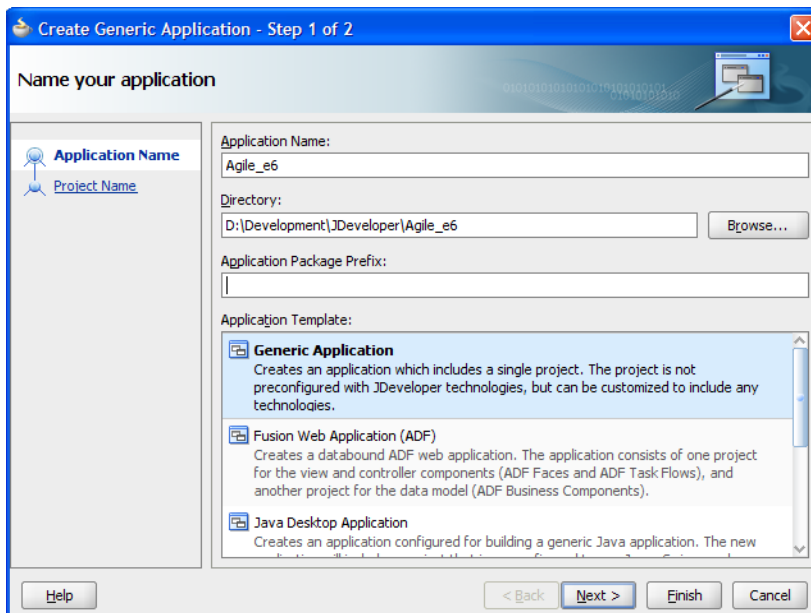
Exercise

1. Select the Application Navigator tab and click New Application (alternatively, you can select File | New to bring up the New Gallery, then select General | Generic Application to create a new application.)

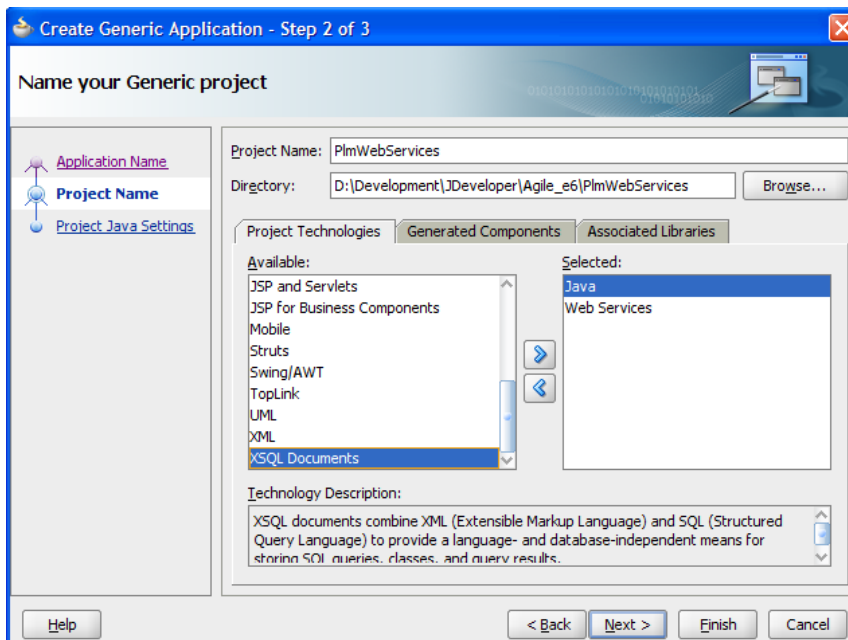


2. In the Create Application dialog box, set the application name to **Agile_e6**.

Note Leave the **Application Package Prefix** property blank.

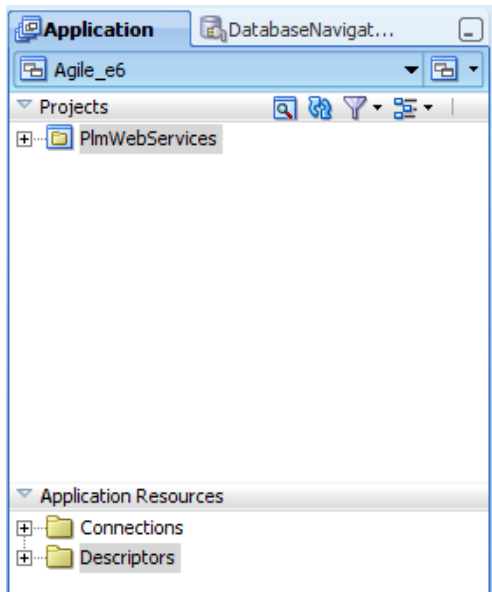


3. Click **Next**.
4. Name the default Project **PImWebServices**.
Each web service scenario will be created in its own project.
This project will be used for the annotation web service scenario.
5. On the Project Technologies tab, shuttle **Web Services** from the Available list to the Selected list (this also selects the Java technology.)



6. Click **Finish**.

The Application Navigator should look like this:

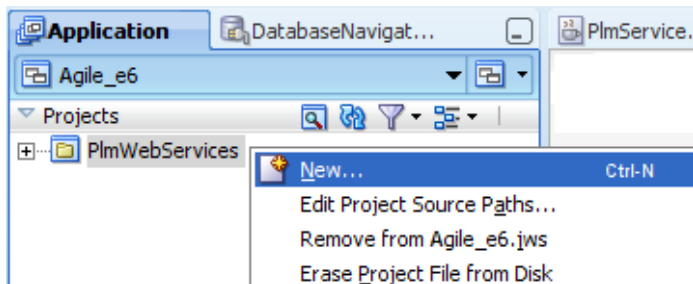


Step 2 : Add a Plain Old Java Object (POJO) to contain the Web Service Method

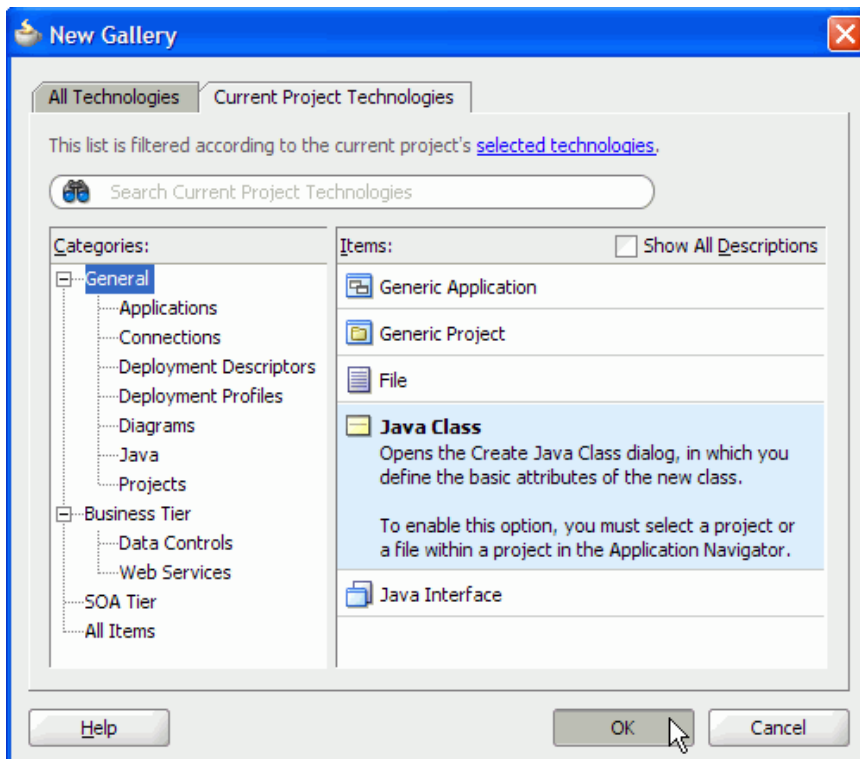
In this section you create a plain Java class containing a method to return an input value with some text. The class is annotated to define it as a web service. It also contains an annotation to define the method as part of the web service. This section shows how to modify the method properties using the Code Editor, Property Inspector, and Structure window. Once compiled and deployed to the integrated server, the web service is then run using the HTTP Analyzer, which returns the result of the method.

Exercise

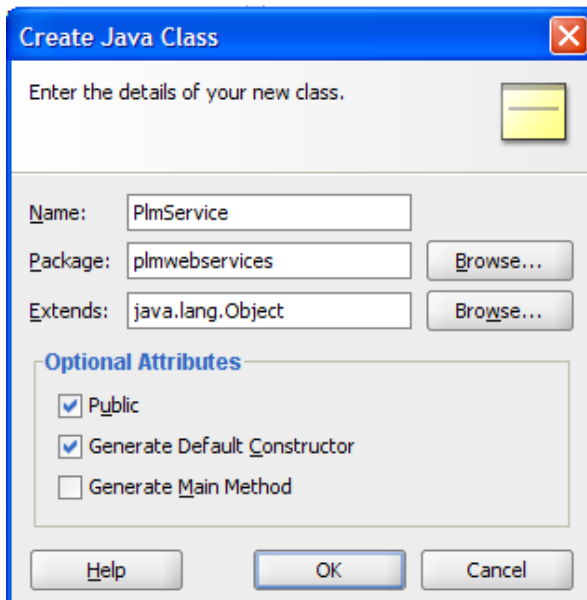
1. Create the Java class from the Application Navigator. Right-click the Annotation project and select **New**.



2. In the New Gallery, select the **General** category, then select the **Java Class** item and click **OK**.



3. Name the class **PlmService** and set the Package to **plmwebservices**. The remaining properties can be left at their default.



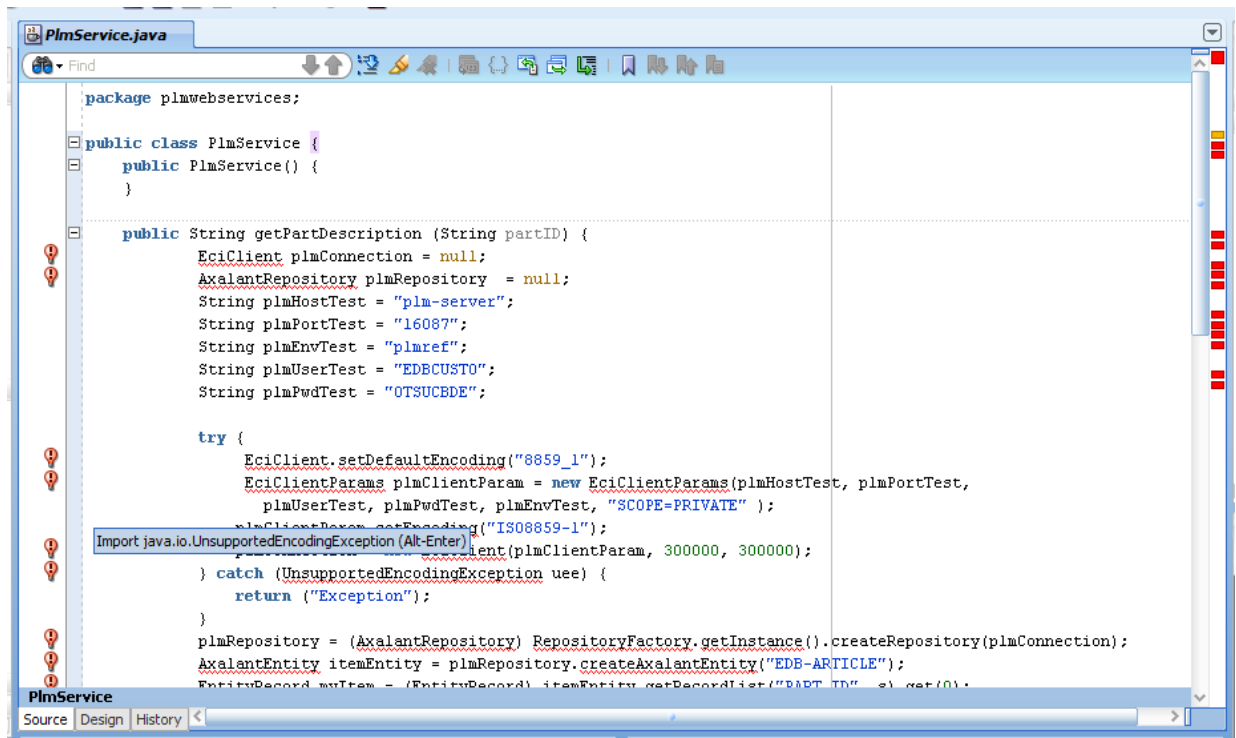
4. Click **OK** to create the class and display it in the Code Editor.
5. Following the class definition in `PlmService`, add a method named `getPartDescription()` that accepts a Part ID string and returns the Part Description.

```
// =====
// Copyright © 2009 Oracle Corporation <http://www.oracle.com/>.
// All rights reserved.
//
// This file, which has been provided by Oracle Corporation as part of
// an Oracle® product for use ONLY by licensed users of the product,
// includes CONFIDENTIAL and PROPRIETARY information of Oracle Corporation.
//
// USE OF THIS SOFTWARE IS GOVERNED BY THE TERMS AND CONDITIONS OF THE LICENSE
// AGREEMENT AND LIMITED WARRANTY FURNISHED WITH THE PRODUCT.
//
// IN PARTICULAR, YOU WILL INDEMNIFY AND HOLD ORACLE CORPORATION, ITS
// RELATED COMPANIES AND ITS SUPPLIERS, HARMLESS FROM AND AGAINST ANY CLAIMS
// OR LIABILITIES ARISING OUT OF THE USE, REPRODUCTION, OR DISTRIBUTION OF
// YOUR PROGRAMS, INCLUDING ANY CLAIMS OR LIABILITIES ARISING OUT OF OR
// RESULTING FROM THE USE, MODIFICATION, OR DISTRIBUTION OF PROGRAMS OR FILES
// CREATED FROM, BASED ON, AND/OR DERIVED FROM THIS SAMPLE SOURCE CODE FILE.
// =====

public String getPartDescription (String partID) {
    EciClient plmConnection = null;
    AxalantRepository plmRepository = null;
    String plmHostTest = "plm-server";
    String plmPortTest = "16087";
    String plmEnvTest = "plmref";
    String plmUserTest = "EDBCUSTO";
    String plmPwdTest = "OTSUCBDE";

    try {
        EciClient.setDefaultEncoding("8859_1");
        EciClientParams plmClientParam = new EciClientParams(plmHostTest,
plmPortTest,
        plmUserTest, plmPwdTest, plmEnvTest, "SCOPE=PRIVATE" );
        plmClientParam.setEncoding("ISO8859-1");
        plmConnection = new EciClient(plmClientParam, 300000, 300000);
    } catch (UnsupportedEncodingException uee) {
        return ("Exception");
    }
    plmRepository = (AxalantRepository)
RepositoryFactory.getInstance().createRepository(plmConnection);
    AxalantEntity itemEntity = plmRepository.createAxalantEntity("EDB-
ARTICLE");
    EntityRecord myItem = (EntityRecord) itemEntity.getRecordList("PART_ID",
partID).get(0);
    String partDescription = myItem.getString("T_MASTER_DAT.PART_NAME_ENG");
    System.out.println("item is "+partDescription);

    if (plmConnection.isConnected())
        plmConnection.close();
    return "Description is " + partDescription;
}
}
```



Then add following Import statements at the beginning of the code:

```
import com.agile.eci.EciClient;
import com.agile.eci.EciClientParams;
import com.agile.eci.EciConParams;
import com.agile.er.EntityRecord;
import com.agile.er.Record;
import com.agile.er.axalant.AxalantRepository;
import com.agile.security.EciTicket;
import com.agile.er.RepositoryFactory;
import com.agile.er.axalant.AxalantEntity;
```

```
import java.io.UnsupportedEncodingException;
```

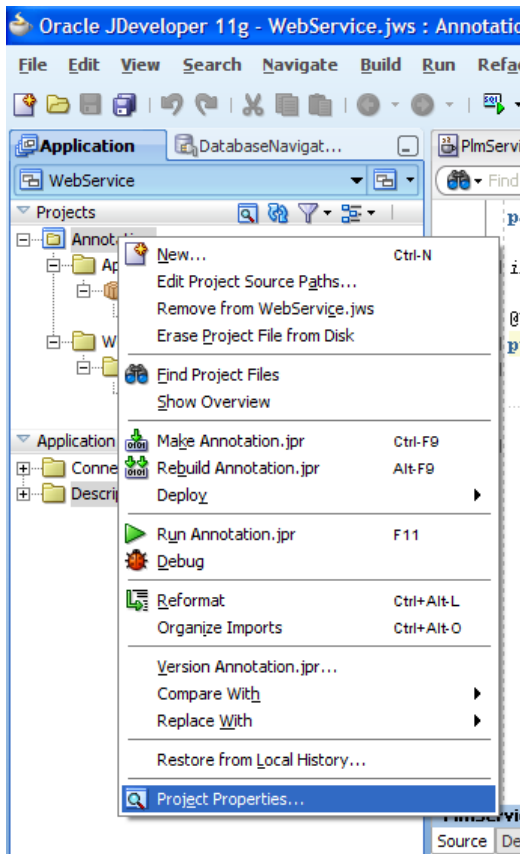
6. **Save** your work.

Note You will get many error messages when pasting above source code because you are using the Agile e6 Java API, but the API libraries are still missing. So, next step is to provide the Java API libraries in JDeveloper:

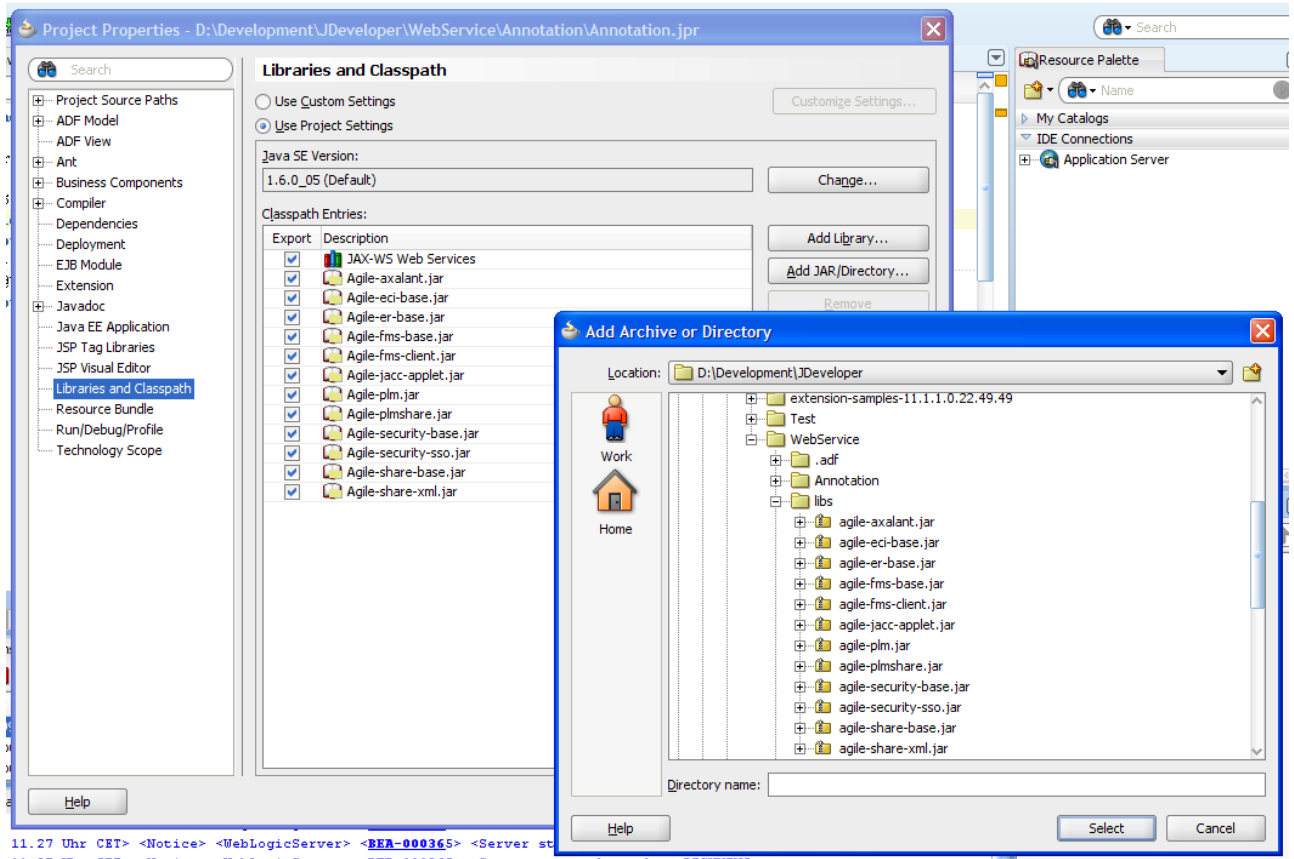
7. In the project directory, add a directory libs and copy all relevant Java API libs into that directory:

Name	Size	Type	Date Modified
agile-axalant.jar	315 KB	Executable Jar File	31.08.2007 11:20
agile-eci-base.jar	293 KB	Executable Jar File	31.08.2007 11:20
agile-er-base.jar	91 KB	Executable Jar File	31.08.2007 11:20
agile-fms-base.jar	47 KB	Executable Jar File	31.08.2007 11:20
agile-fms-client.jar	45 KB	Executable Jar File	31.08.2007 11:20
agile-jacc-applet.jar	41 KB	Executable Jar File	31.08.2007 11:20
agile-plm.jar	92 KB	Executable Jar File	31.08.2007 11:20
agile-plmshare.jar	23 KB	Executable Jar File	31.08.2007 11:20
agile-security-base.jar	56 KB	Executable Jar File	31.08.2007 11:20
agile-security-ssso.jar	20 KB	Executable Jar File	31.08.2007 11:20
agile-share-base.jar	633 KB	Executable Jar File	31.08.2007 11:20
agile-share-xml.jar	49 KB	Executable Jar File	31.08.2007 11:20

- Then you need to make JDeveloper aware of the new libraries. Go to the Project settings and add these libraries for this project:

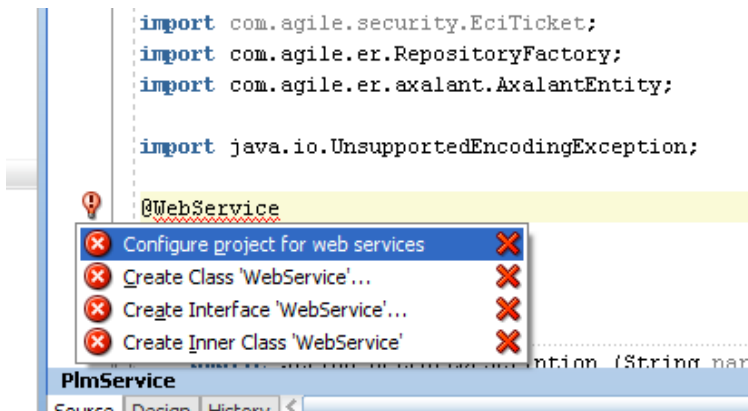


- In the Project Properties, go to *Libraries and Classpath* and add the libraries which you copied to the *libs* directory before via the button *Add JAR/Directory*:

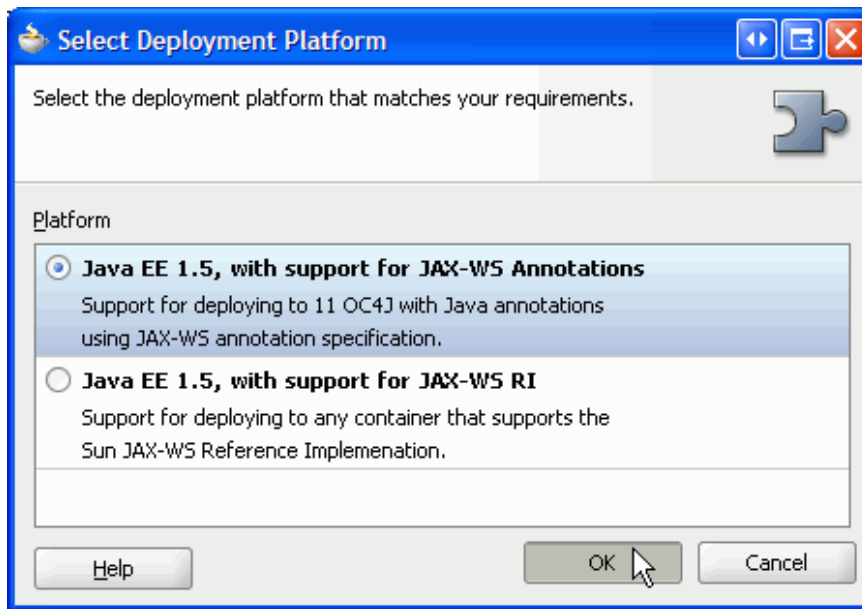


Now all lines in your source code, which showed an error because of the missing libraries, should show normal and no compilation error should come up anymore.

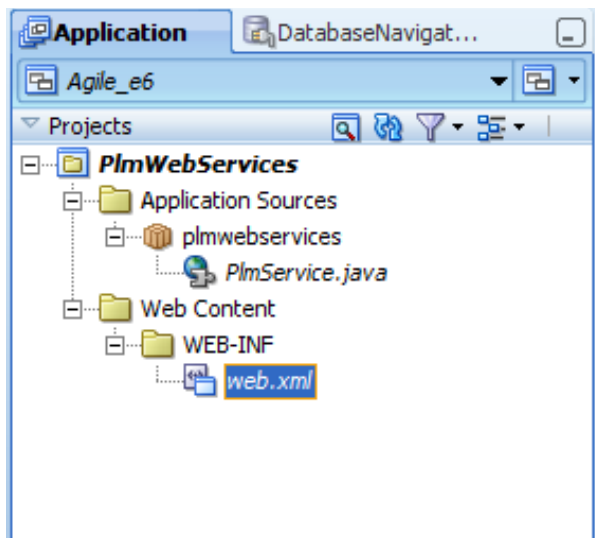
10. While still in the class, add an `@WebService` annotation before the class definition. This annotation denotes that the class contains a method to be used by a web service.
11. In the margin, click the **Quick Hint** (light bulb icon) and select **Configure project for web services**.



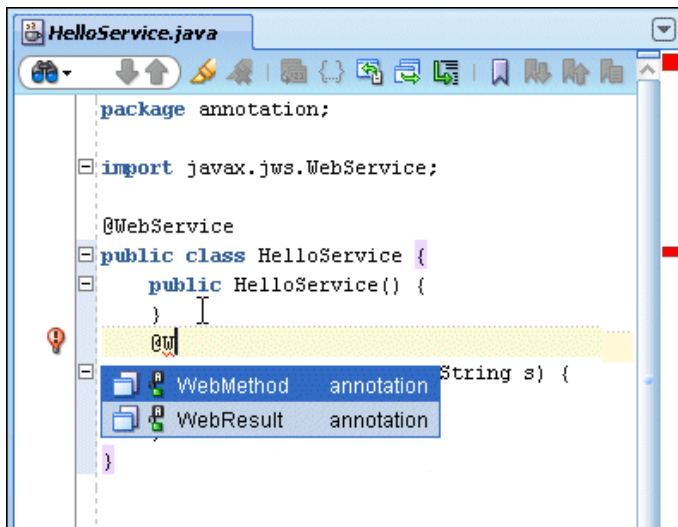
12. In the Select Deployment Platform dialog box, ensure that Java EE 1.5, with support for JAX-WS Annotations is selected, and then click OK.



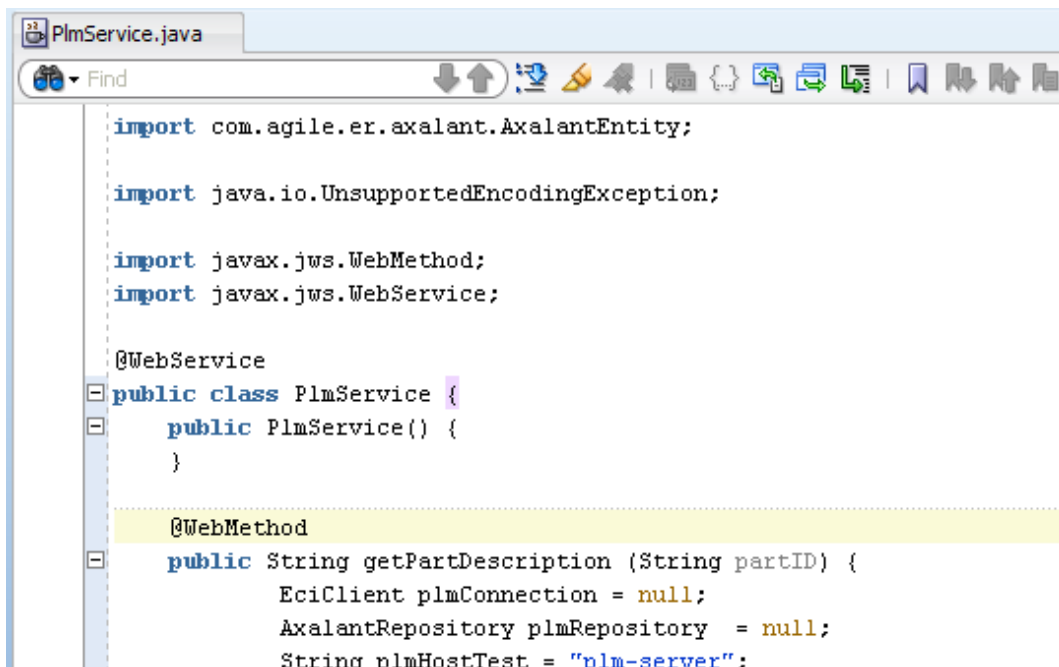
This step adds the `javax.jws.WebService` import statement to the Java class and creates a `web.xml` file.



13. Back in the class, create a second annotation before the `getPartDescription()` method. The annotation signifies this is the method to be exposed from the web service. Add a blank line above the `getPartDescription()` method, and start typing `@WebMethod`. A popup shows you the available syntaxes. Select `WebMethod`.



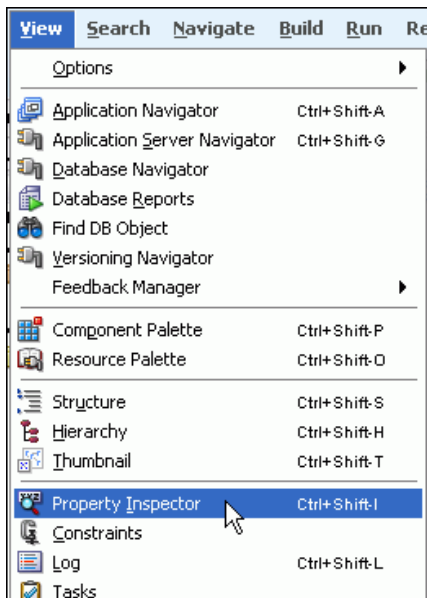
14. Add the import javax.jws.WebMethod; statement. The class should look like the image below.



15. Save your work.

You can use the Property Inspector to modify the characteristics of the class. In the menu bar, select **View | Property Inspector** and it will open as a tab in the bottom portion of the IDE.

Note: If the Property Inspector opens in a different part of the IDE, you can drag its tab and drop it on the bottom panel if you would rather work with it there.



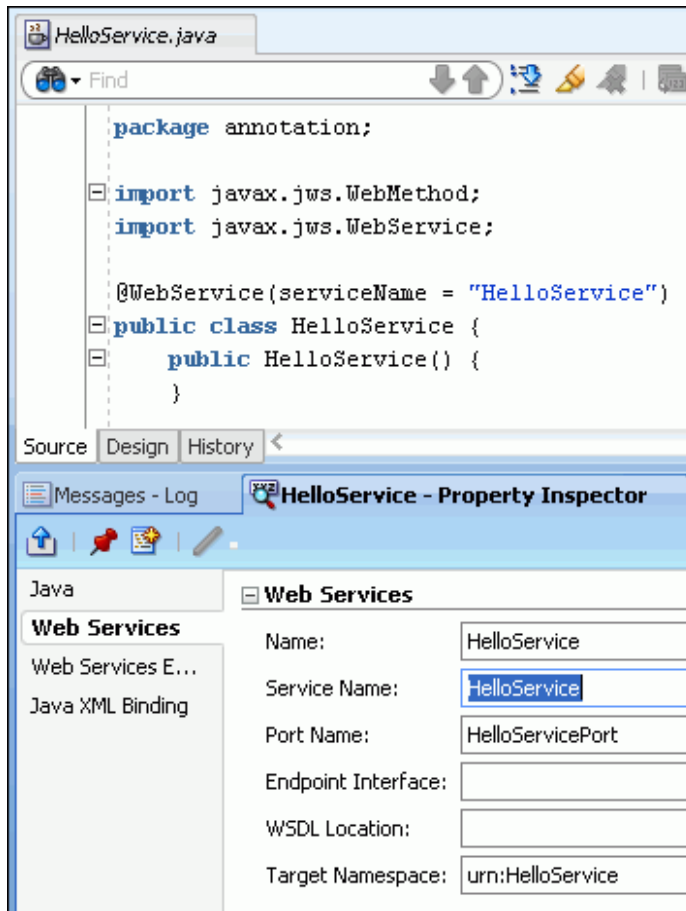
- To display the properties in the Property Inspector, select the Source tab at the bottom of the Structure window, then select the top level PlmService class name



- The Property Inspector displays a few finger tabs on the left side of the window (these could be expandable headings if the Property Inspector window is large enough.) Select the Web Services tab and notice that the Service Name has the word 'Service' appended to the class name. If you don't want to have the service named "PlmServiceService", you can change it,

and the class reflects the change.

18. Change the Service Name to PlmService and save your work.



In the Code Editor you can see that the `@WebService` annotation updated to reflect the new service name. Conversely, changes in the Code Editor, are synchronized in the Property Inspector. This functionality is available at the method level, too.

You now have a class, defined as a web service, that contains an exposed method. In the next section you test the web service.

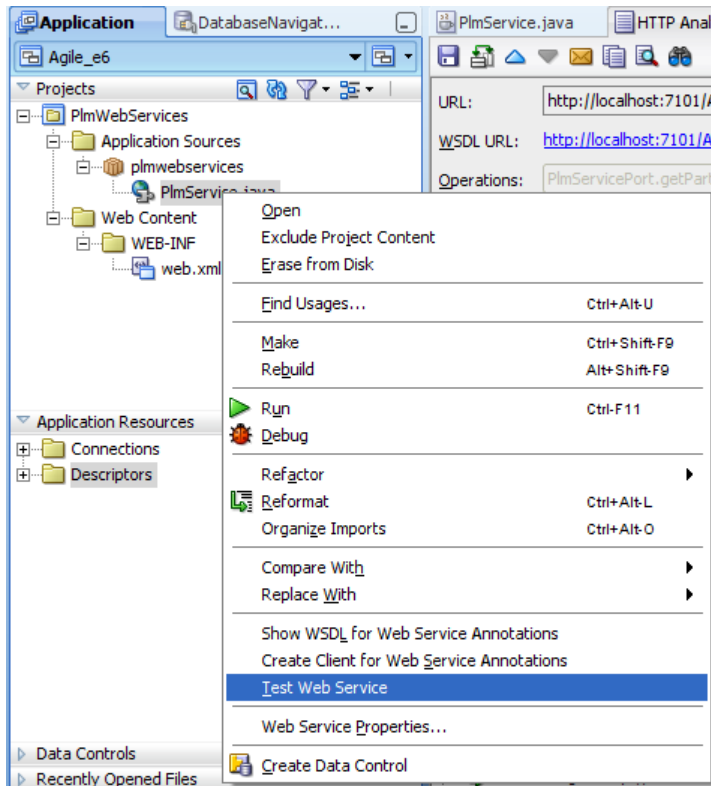
Step 3: Test the Web Service

In this section you compile, deploy and test the web service. The HTTP Analyzer is the testing mechanism for web services. When you use the HTTP analyzer to test web services, the service is compiled and deployed to the integrated server. The analyzer is then invoked, enabling you to send and receive values from the web service.

Exercise

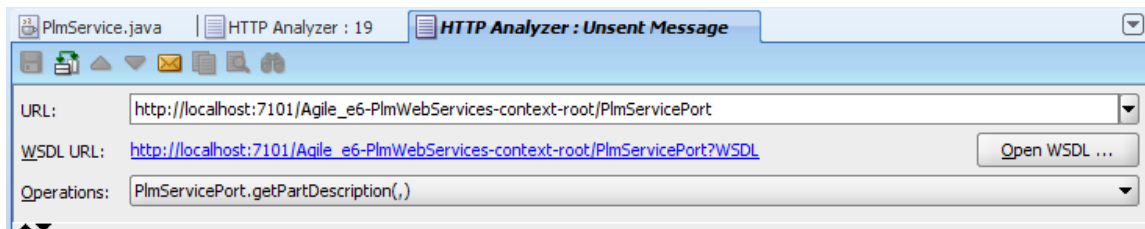
1. Before testing the web service, check that your web browser settings are correct.

1. Choose **Tools > Preferences** and then scroll down the list on the left to select the **Web Browser and Proxy** page.
2. Ensure that the **Use HTTP Proxy Server** check box is *not* selected, then click **OK**.
2. In the Application Navigator, right-click the PlmService.java node and in the context menu, select **Test Web Service**.

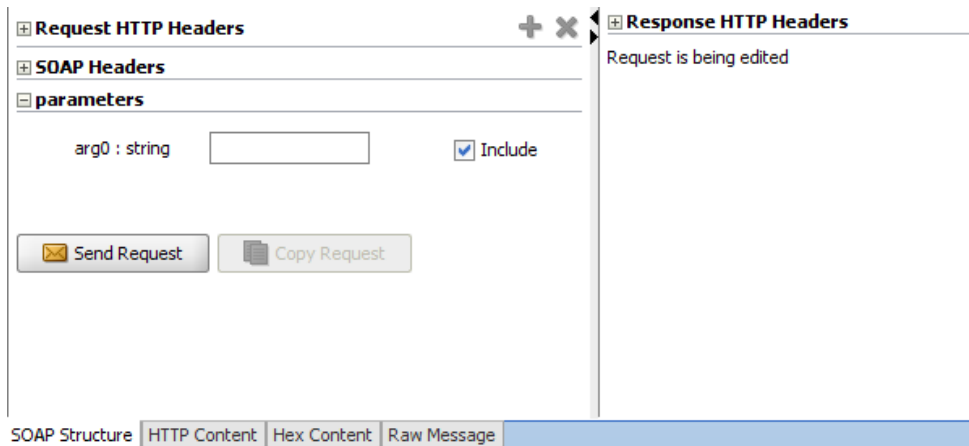


This option invokes the integrated server, deploys the service and then starts the analyzer. It may take a few seconds to start the integrated server if it is being run for the first time. If this is the first time you test a service, Windows may ask you about blocking content. Allow the content to be displayed.

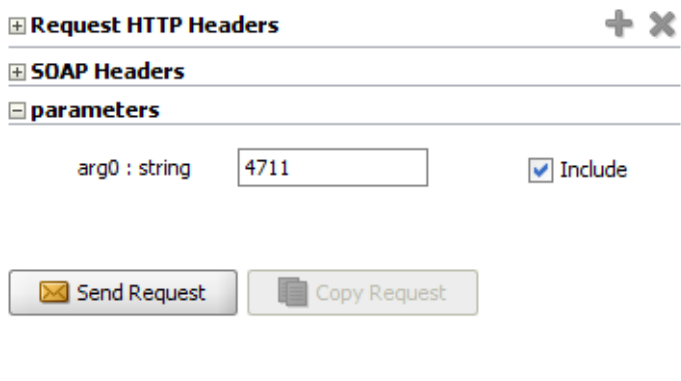
3. The top portion of the HTTP Analyzer editor window displays the URL for the web service, the WSDL URL, and the exposed Operations.



The bottom portion of the analyzer is split into two areas: Request and Response. The request area shows all the arguments from the exposed method. When the web service is executed, the Response area shows the results.



- In the Request area, enter <part id> in the arg0 field.



- In the top area of the analyzer, click Send Request, or click the Send Request button



The analyzer sends the request to the service, and after a few seconds the return parameter is displayed at the bottom of the Response area, which is the Description of respective part, which we send to the Web Service as input parameter

