



How to integrate Oracle BI Publisher via Web Services in Oracle Forms

(Version 2.2)

November 2008

Authors: **Jürgen Menge**
 Rainer Willems

Contributors: **Mireille Duroussaud**

ORACLE®

This paper describes in a step-by-step way the generation of a client for the Web Services provided by Oracle BI Publisher and how to import and use this client in Oracle Forms.

Used Versions for this example

- Oracle Forms 10.1.2.3
- Oracle JDeveloper 10.1.3.4
<http://www.oracle.com/technology/software/products/jdev/htdocs/soft10134.html>
- BI Publisher Enterprise 10.1.3.4 on WebLogic Server 10.3
(but should also work on OC4J in the same way)

As a step by step example to use Web Services in Oracle Forms see also the Tutorial **SOA for Forms and 4GL Developers: Calling a Web service from Oracle Forms 10.1.2** (http://www.oracle.com/technology/products/forms/htdocs/10gr2/howto/webservicefromforms/ws_1013_from_forms.html)

Oracle BI Publisher Web Service API

With Oracle BI Publisher 10.1.3.3.1 a Public Web Service API was firstly introduced into the product. Compared to the previously existing web service interface (which was not a public one) there were changes in names and parameters of web service method.

A description can be found in the official documentation

http://download.oracle.com/docs/cd/E12844_01/doc/bip.1013/e10416/bip_webservice_101331.htm#CHDGIJHH

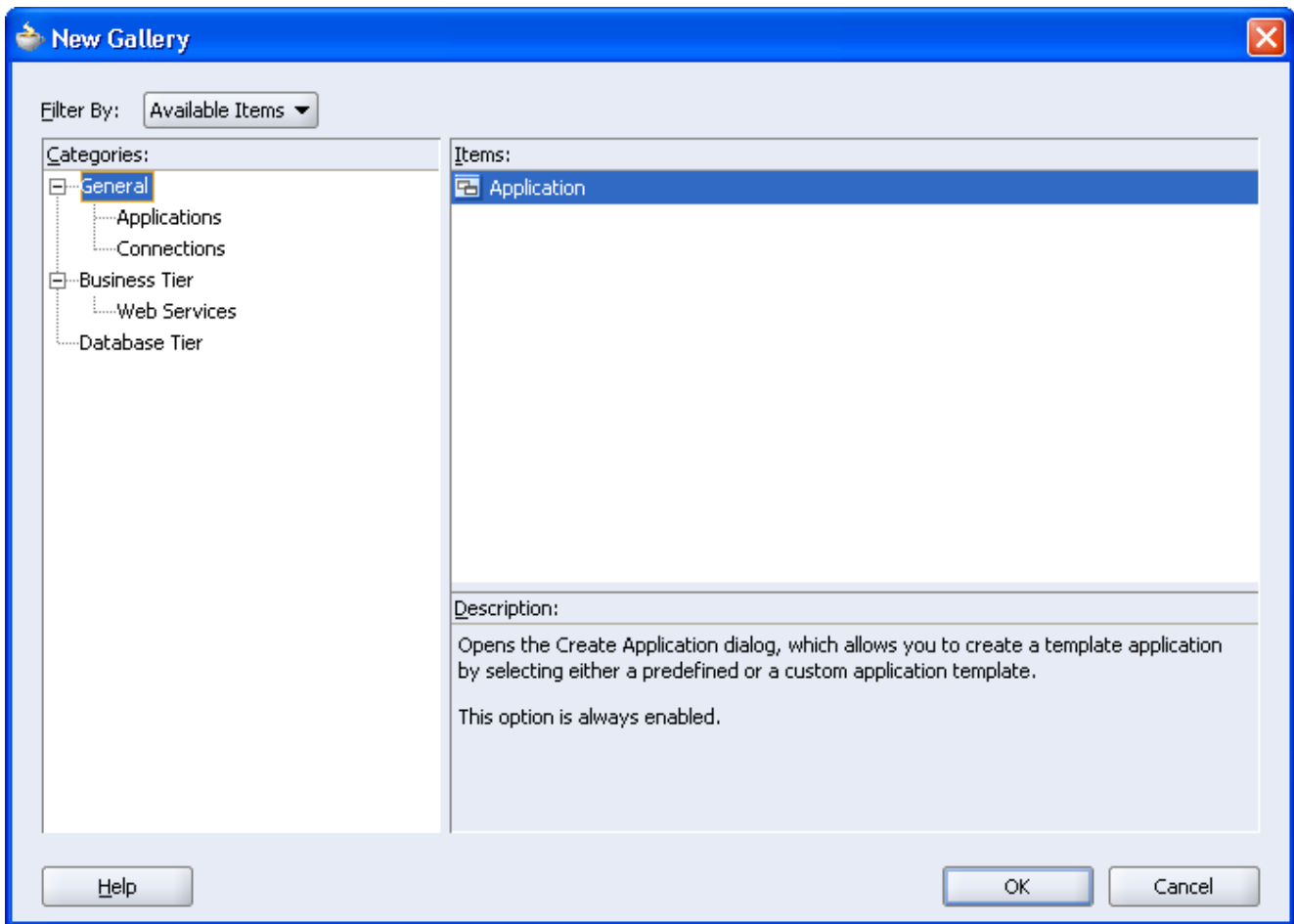
The web services were created by using Apache Axis 1.3.

The WSDL for the web service can be found at:

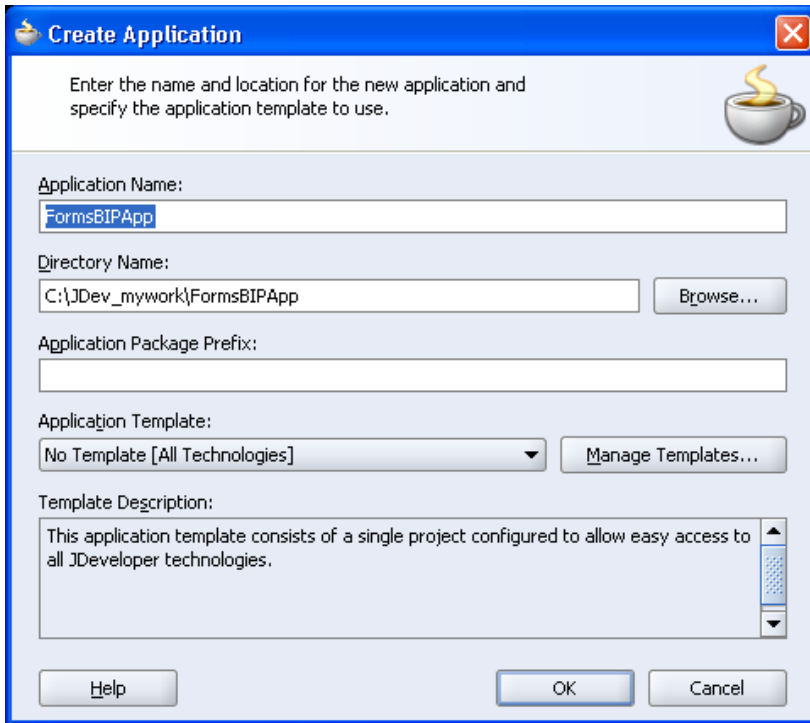
<http://<host>:<port>/xmlpserver/services/PublicReportService?wsdl>

Creating a client for the Web Services with Oracle JDeveloper

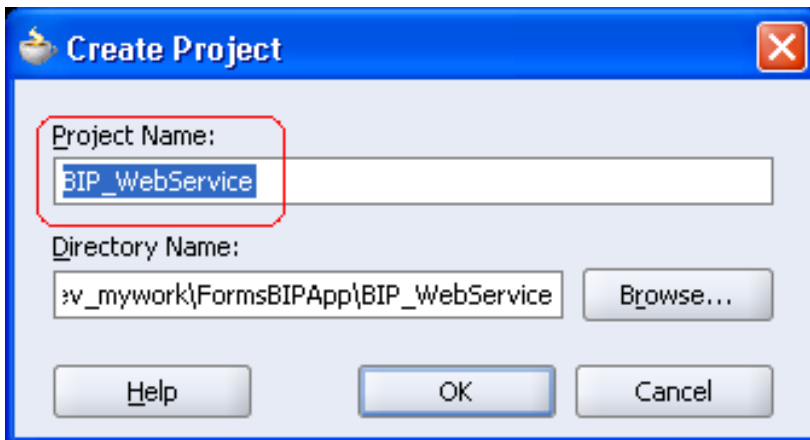
Create a new Application in JDeveloper



Select **No Template** as Application Template



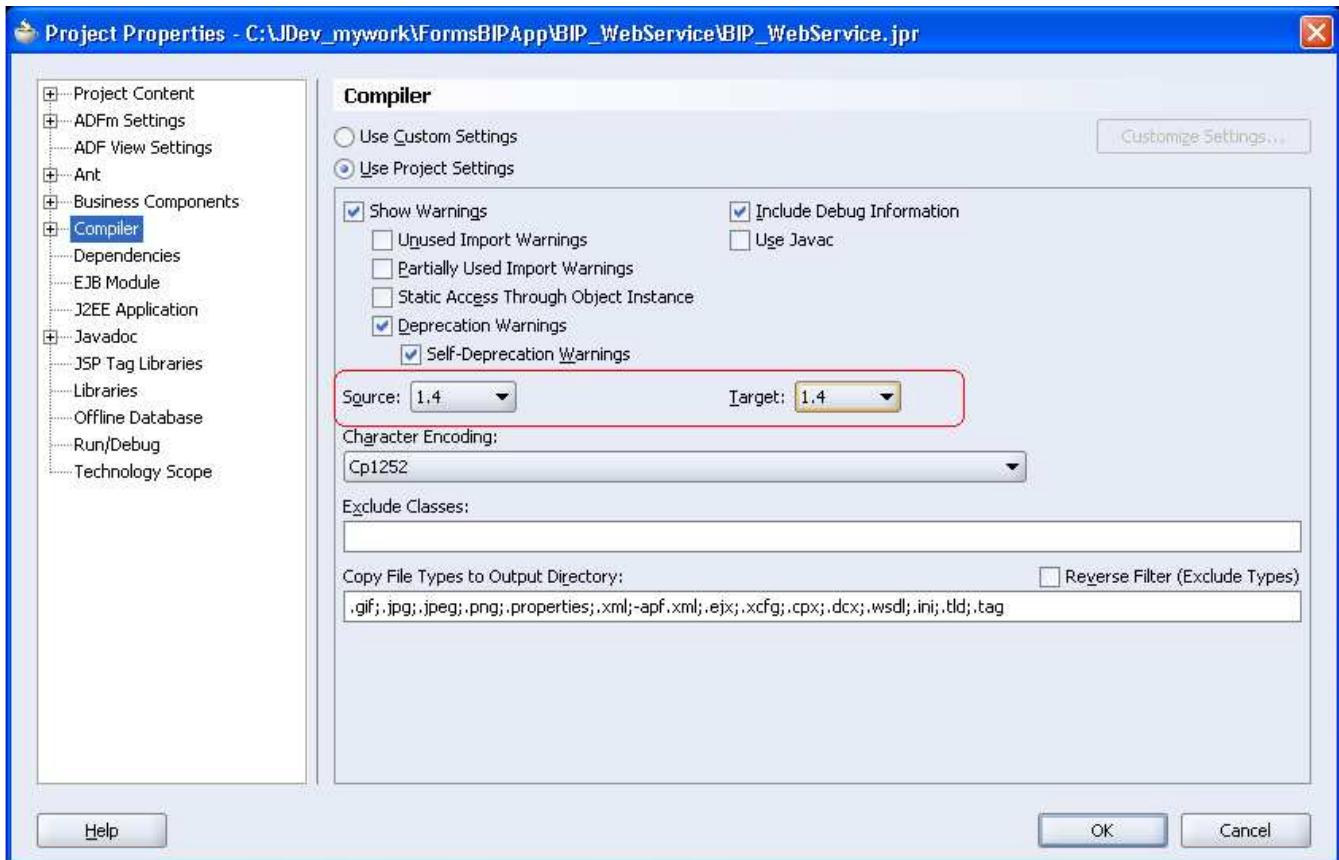
Use **BIP_WebService** as Project Name



When choosing another name pay attention in the document. There are several code pieces using this project name.

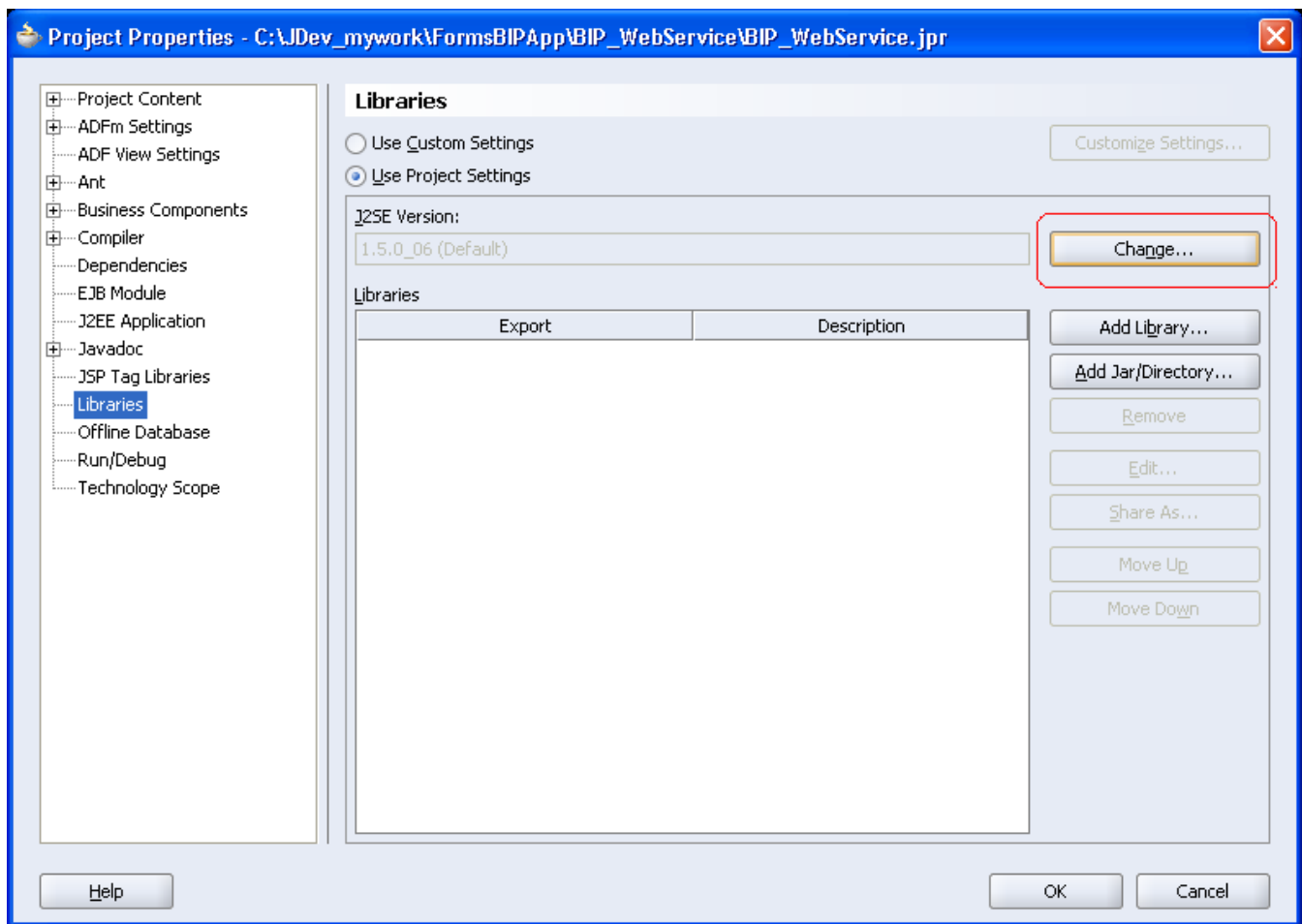
Select the created Project **BIP_WebService.jpr** in the Navigator and choose the **Project Properties** with a right-click.

Select the **Compiler** node and set the **Source** and **Target** dropdown lists to **1.4**. (That's because of the used JRE of Forms 10.1.2)

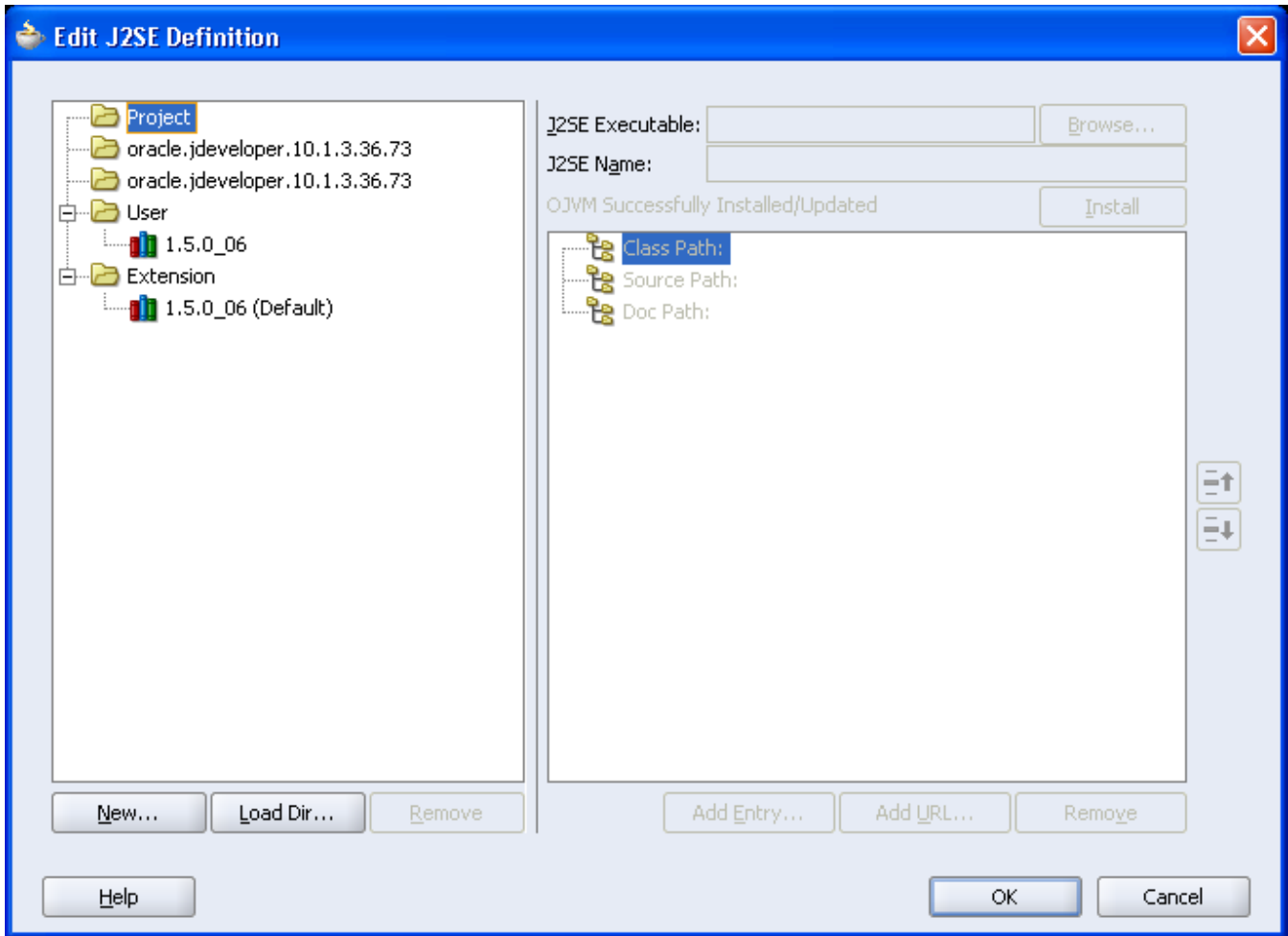


If you had to go via a HTTP-Proxy define this in the Tools-Menu under **Preferences** at **Web Browser and Proxy**. Stay in the project properties.

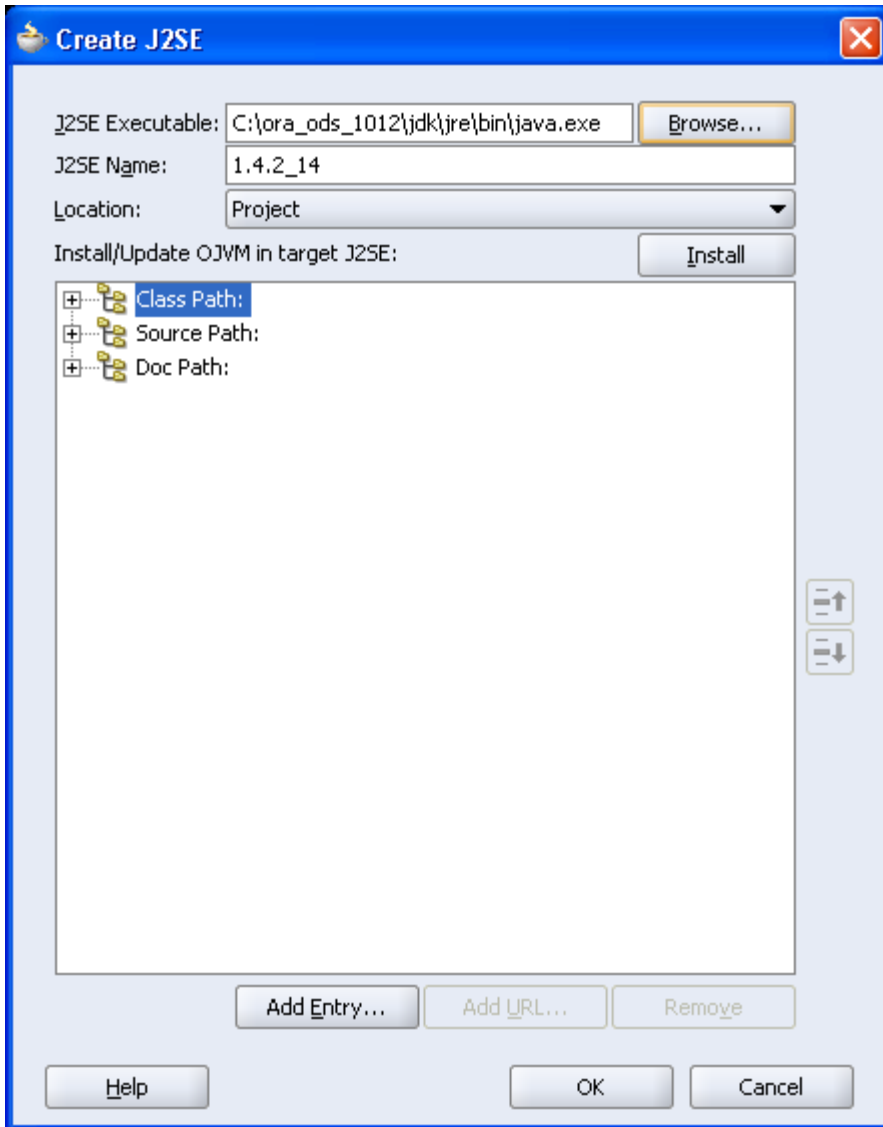
When working with Oracle Forms you have to replace the version of JDK which is used in the project. Select the **Libraries** node in the Project Properties and press the button to **Change** the J2SE version.



In the **Edit J2SE Definition** Window go to the left pane and select the node **Project**.

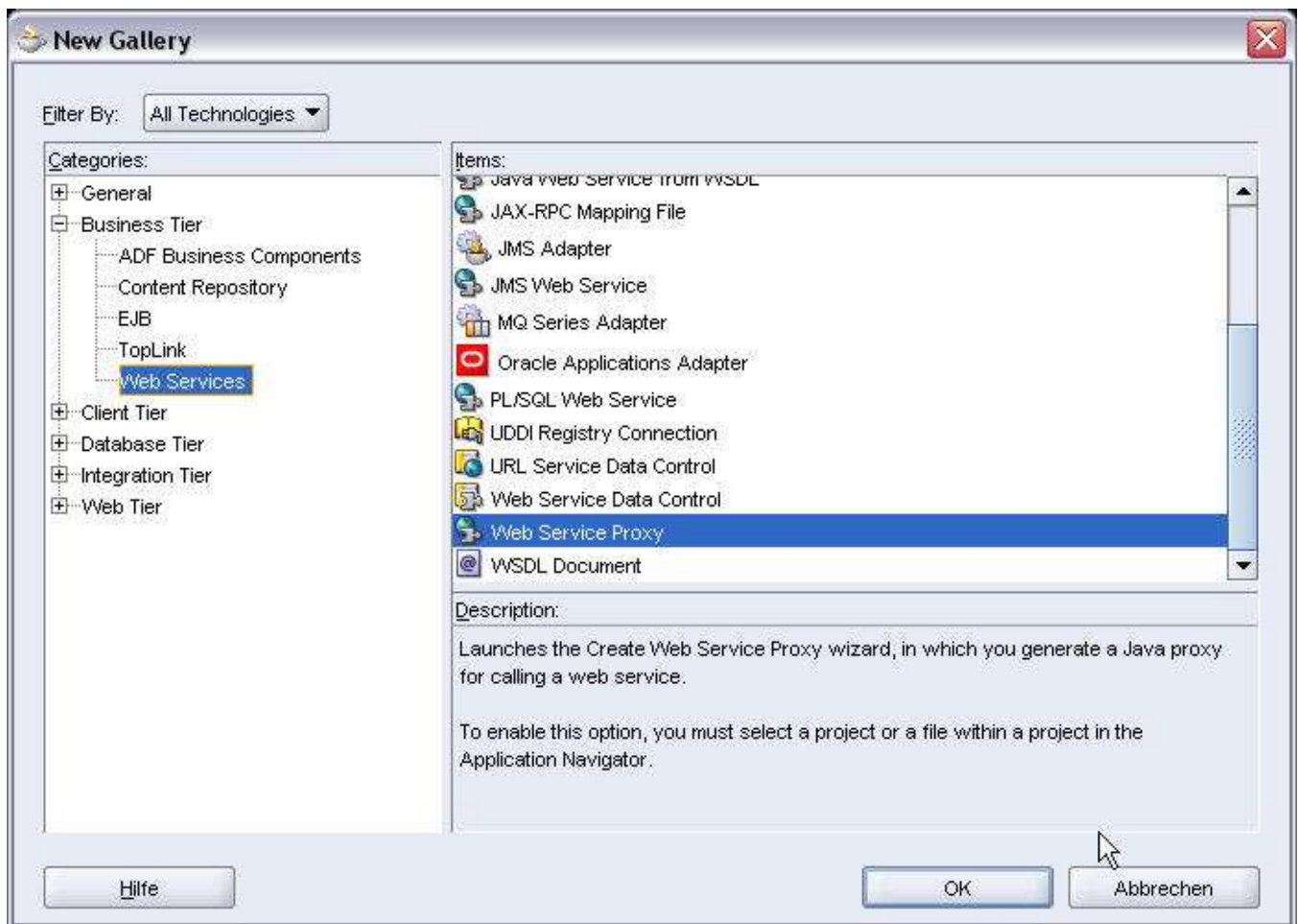


By pressing the button **New** a window **Create J2SE** opens where you can reference the necessary J2SE. Press the button **Browse** and select the file *java.exe* from the directory *\jdk\jre\bin* in the Oracle Home of your Developer Suite.

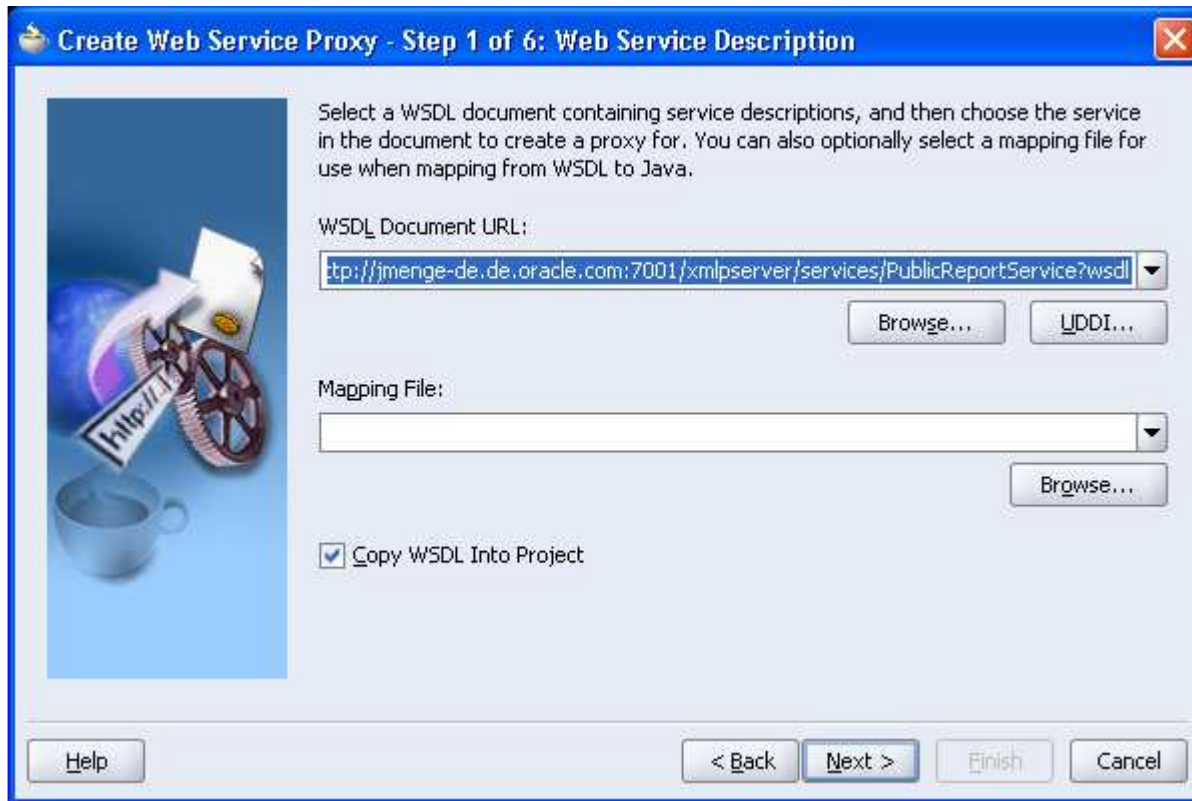


If you submit your selection with **OK** you will see that several jar-Files were added to the classpath. In the window **Project Properties** the version of J2SE should now be set to 1.4.2_14.

After saved the project, right-click the project, choose New and select the Item **Web Services Proxy** in the Category **Business Tier – Web Services**



Choose as your WSDL Document URL the appropriate URL of your BI Publisher Server:
<http://<machine>:<port>/<yourappname>/services/PublicReportService?wsdl>
(Default for <yourappname> is xmlpserver)



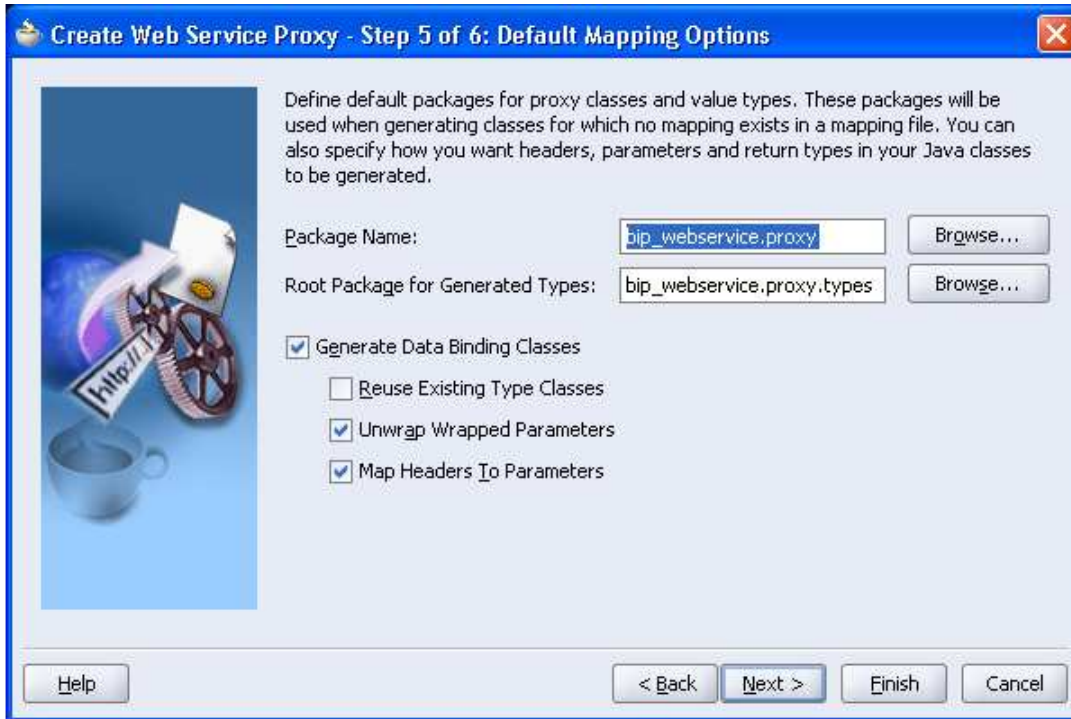
If you run into problems when generating the WSDL from the above URL you could manually create a WSDL file and use it with the second option *Mapping File*.

A complete definition of the WSDL can be found in the documentation at:

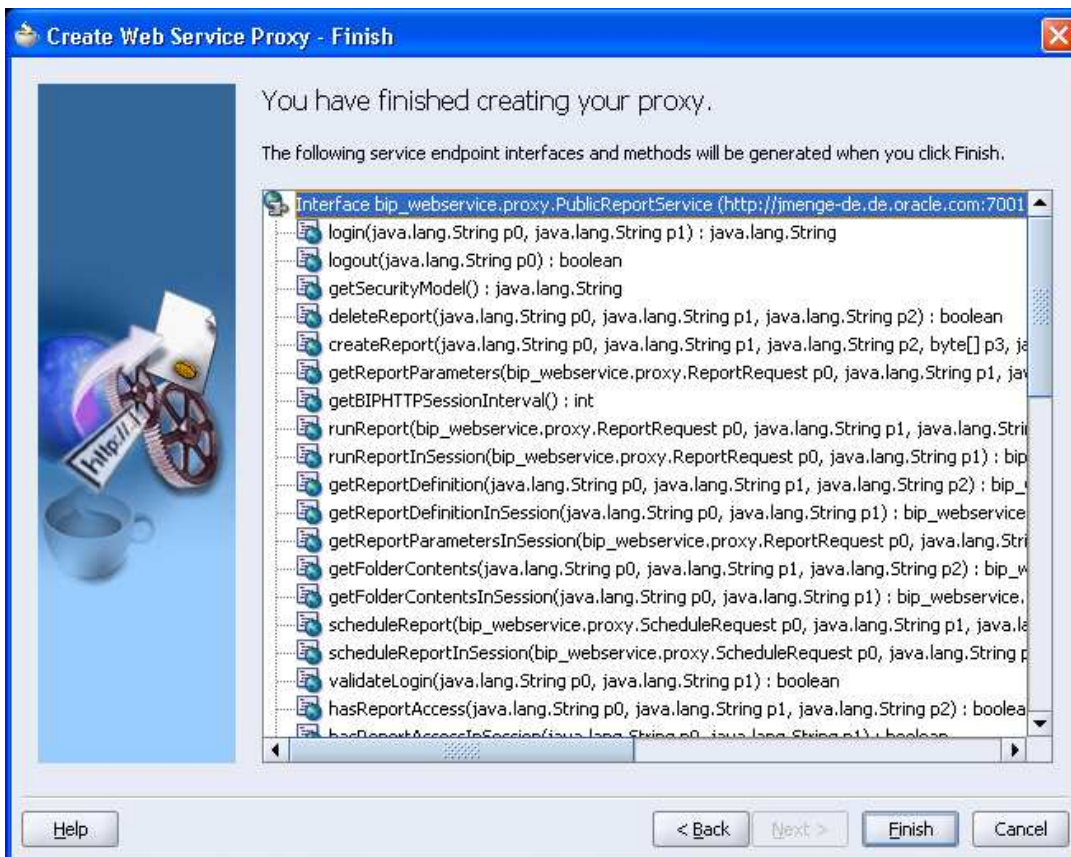
http://download.oracle.com/docs/cd/E12844_01/doc/bip.1013/e10416/bip_websevice_101331.htm#BABHEHEJ

Just copy the text in a file and add it to the content of your project.

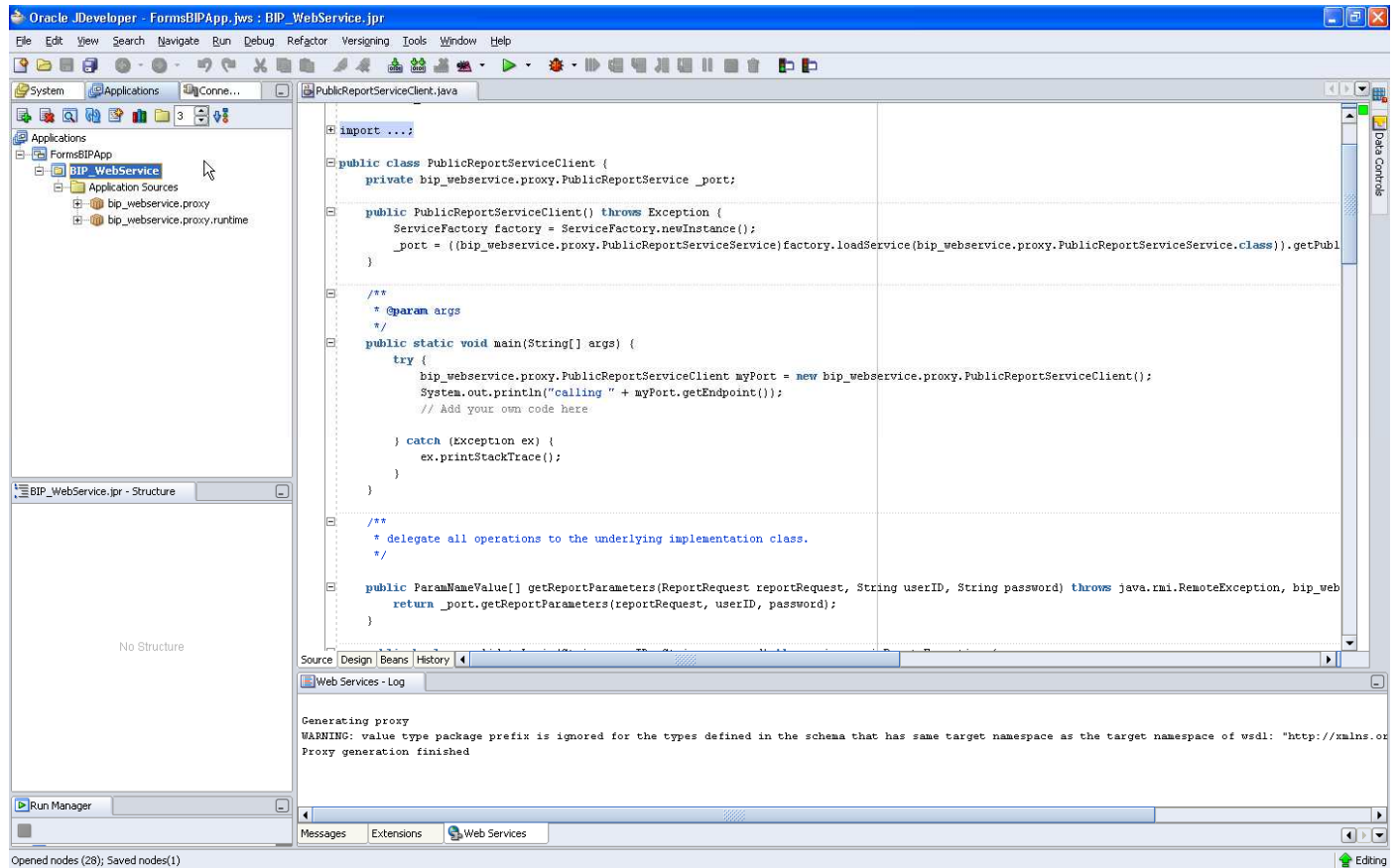
Accept the following two screens and then set your wished **Package Name** (here **bip_webservice.proxy**).



Accept again the two following screens.
Check that in the Finish screen the method **runReport** is available in the service endpoint.



Finally you've created the proxy stub.



Now we will test the web service. Edit the method **main** in the class **PublicReportServiceClient** (find the line `// Add your code here`) with the code you want.

In the documentation there is a test client for all methods of the web service which you can take as an example:

http://download.oracle.com/docs/cd/E12844_01/doc/bip.1013/e10416/bip_webservice_101331.htm#CBADBFAF

For example choose something like the following code fragment to call one of your BI Publisher reports and create it in your preferred output format in the file system.

The source code of the report *Employees.xdo* can be found in Appendix A.

```
public static void main(String[] args) {
    try {
        bip_webservice.proxy.PublicReportServiceClient myPort =
        new bip_webservice.proxy.PublicReportServiceClient();
        System.out.println("calling " + myPort.getEndpoint());
        // Add your own code here
        final String username = "Administrator";
        final String password = "Administrator";
        final String reportAbsolutePath = "/Web Service Test/Employees/Employees.xdo";

        // Testing runReport
        System.out.println("Testing runReport Service");
        ReportRequest repRequest = new ReportRequest();
    }
}
```

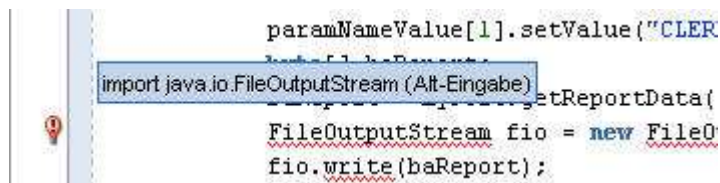
```
repRequest.setReportAbsolutePath(reportAbsolutePath);
repRequest.setAttributeTemplate("Simple");
repRequest.setAttributeFormat("pdf");
repRequest.setAttributeLocale("en-US");
repRequest.setSizeOfDataChunkDownload(-1);
```

```
ParamNameValue[] paramNameValue = new ParamNameValue[2];
paramNameValue[0] = new ParamNameValue();
paramNameValue[0].setName("p_deptno");
paramNameValue[0].setValues(new String[] {"20"});
paramNameValue[1] = new ParamNameValue();
paramNameValue[1].setName("p_job");
paramNameValue[1].setValues(new String[] {"CLERK"});
repRequest.setParameterNameValues(paramNameValue);
```

```
ReportResponse repResponse = new ReportResponse();
repResponse = myPort.runReport(repRequest,username,password);
String contentType = repResponse.getReportContentType();
System.out.println(contentType);
```

```
byte[] baReport = repResponse.getReportBytes();
FileOutputStream fio = new FileOutputStream("C:\\temp\\test.pdf");
fio.write(baReport);
fio.close();
```

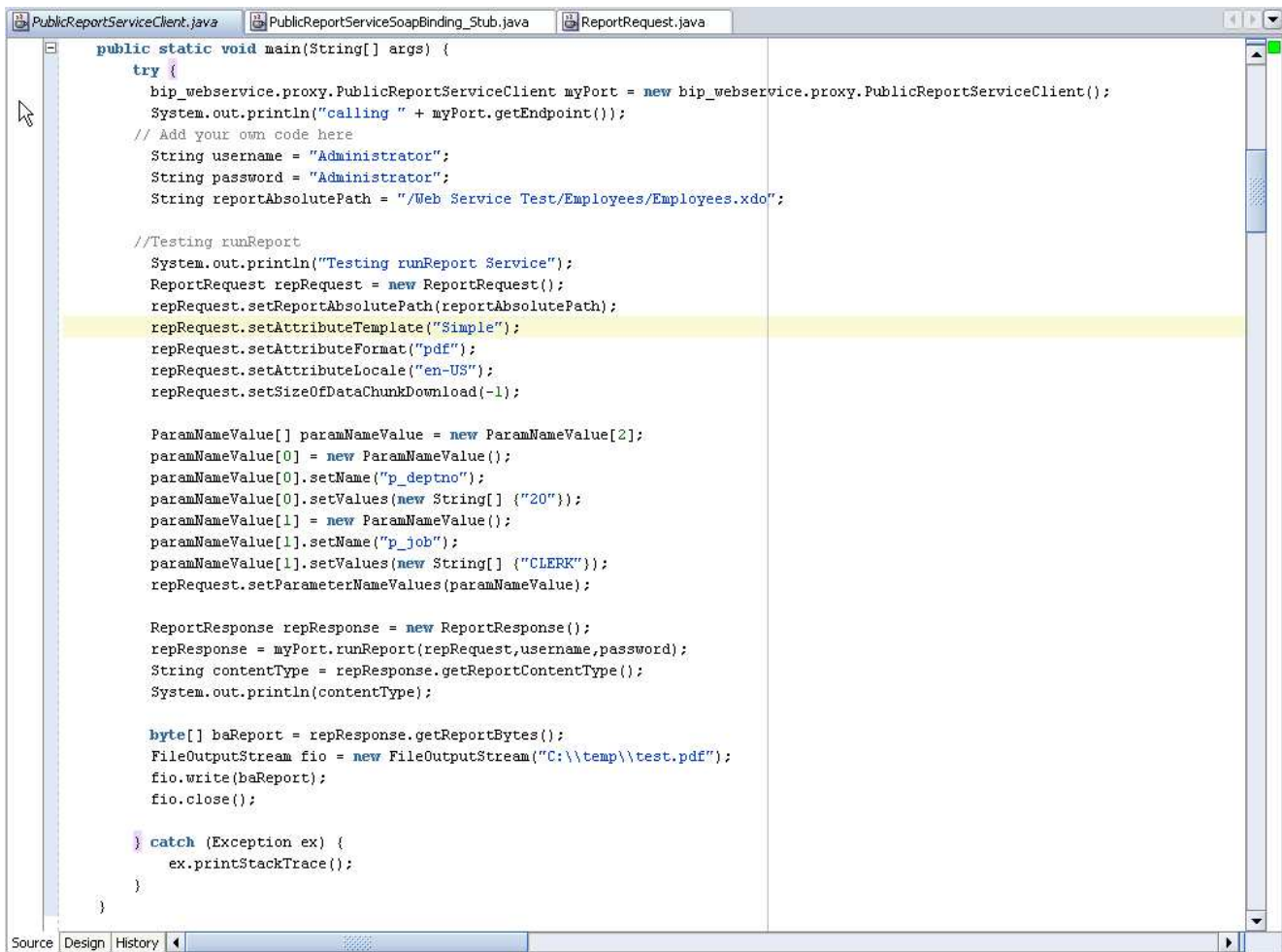
After following JDevelopers hint to *import java.io.FileOutputStream*

A screenshot of an IDE window showing a code snippet. The code is as follows:

```
paramNameValue[1].setValue("CLERK");
// ...
import java.io.FileOutputStream (Alt-Eingabe);
// ...
FileOutputStream fio = new FileOutputStream("C:\\temp\\test.pdf");
fio.write(baReport);
```

The line `import java.io.FileOutputStream (Alt-Eingabe);` is highlighted with a blue selection box. To the left of the code, there is a vertical toolbar with a lightbulb icon, indicating a hint or suggestion.

you've the following code:



```
public static void main(String[] args) {
    try {
        bip_webservice.proxy.PublicReportServiceClient myPort = new bip_webservice.proxy.PublicReportServiceClient();
        System.out.println("calling " + myPort.getEndpoint());
        // Add your own code here
        String username = "Administrator";
        String password = "Administrator";
        String reportAbsolutePath = "/Web Service Test/Employees/Employees.xdo";

        //Testing runReport
        System.out.println("Testing runReport Service");
        ReportRequest repRequest = new ReportRequest();
        repRequest.setReportAbsolutePath(reportAbsolutePath);
        repRequest.setAttributeTemplate("Simple");
        repRequest.setAttributeFormat("pdf");
        repRequest.setAttributeLocale("en-US");
        repRequest.setSizeOfDataChunkDownload(-1);

        ParamNameValue[] paramNameValue = new ParamNameValue[2];
        paramNameValue[0] = new ParamNameValue();
        paramNameValue[0].setName("p_deptno");
        paramNameValue[0].setValues(new String[] {"20"});
        paramNameValue[1] = new ParamNameValue();
        paramNameValue[1].setName("p_job");
        paramNameValue[1].setValues(new String[] {"CLERK"});
        repRequest.setParameterNameValues(paramNameValue);

        ReportResponse repResponse = new ReportResponse();
        repResponse = myPort.runReport(repRequest,username,password);
        String contentType = repResponse.getReportContentType();
        System.out.println(contentType);

        byte[] baReport = repResponse.getReportBytes();
        FileOutputStream fio = new FileOutputStream("C:\\temp\\test.pdf");
        fio.write(baReport);
        fio.close();

    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

Starting with BI Publisher 10.1.3.4 there is a new and important parameter in the type ReportRequest - **SizeOfDataChunkDownload**. According to the documentation it should be set to **-1** if you don't want to chunk the resulting data. Otherwise BI Publisher will produce a correct outfile but you will never see anything from it in your resulting document. If you use an earlier version of BI Publisher you can skip this parameter.

```

PublicReportServiceClient.java | PublicReportServiceSoapBinding_Stub.java | ReportRequest.java
* @param args
*/
public static void main(String[] args) {
    try {
        bip_webservice.proxy.PublicReportServiceClient myPort = new bip_webservice.proxy.PublicReportServiceClient
        System.out.println("calling " + myPort.getEndpoint());
        // Add your own code here
        String username = "Administrator";
        String password = "Administrator";
        String reportAbsolutePath = "/Web Service Test/Employees/Employees.xdo";

        //Testing runReport
        System.out.println("Testing runReport Service");
        ReportRequest repRequest = new ReportRequest();
        repRequest.setReportAbsolutePath(reportAbsolutePath);
        repRequest.setAttributeTemplate("Simple");
        repRequest.setAttributeFormat("pdf");
        repRequest.setAttributeLocale("en-US");
        repRequest.setSizeOfDataChunkDownload(-1);

        ParamNameValue[] paramNameValue = new ParamNameValue[2];
        paramNameValue[0] = new ParamNameValue();
        paramNameValue[0].setName("p_deptno");
        paramNameValue[0].setValues(new String[] {"20"});
        paramNameValue[1] = new ParamNameValue();
        paramNameValue[1].setName("p_job");
        paramNameValue[1].setValues(new String[] {"CLERK"});
        repRequest.setParameterNameValues(paramNameValue);

        ReportResponse repResponse = new ReportResponse();
        repResponse = myPort.runReport(repRequest,username,password);
        String contentType = repResponse.getReportContentType();
        System.out.println(contentType);

        byte[] baReport = repResponse.getReportBytes();
        FileOutputStream fio = new FileOutputStream("C:\\temp\\test.pdf");
        fio.write(baReport);
        fio.close();
    }
}
Source | Design | History

```

You can test your Web Service Proxy by clicking RUN (from the context menu of *PublicReportServiceClient.java* or in the toolbar. During the test the method `main()` is used.

To call the method `runReport()` from our Forms application we need some additional code because we should not:

- use the method `main()` from outside
- modify the generated method `runReport()`.

There are two conceptual ways to do that:

- write a wrapper class around *PublicReportServiceClient.java*
- write an additional customized method in *PublicReportServiceClient.java* which we will call from outside

For reasons of simplicity we'll go the second way and add a method `callRunReport()` in *PublicReportServiceClient.java*.

First we define the parameters in the class *PublicReportServiceClient*:

```

public class ReportServiceClient {
    private bip_webservice.proxy.PublicReportService _port;
    private String username;
    private String password;
}

```

```

private String reportPath;
private String format;
private String template;
private String locale;
private int sizeOfDataChunkDownload;
private String paramName;
private String paramValue;
private String outFile;

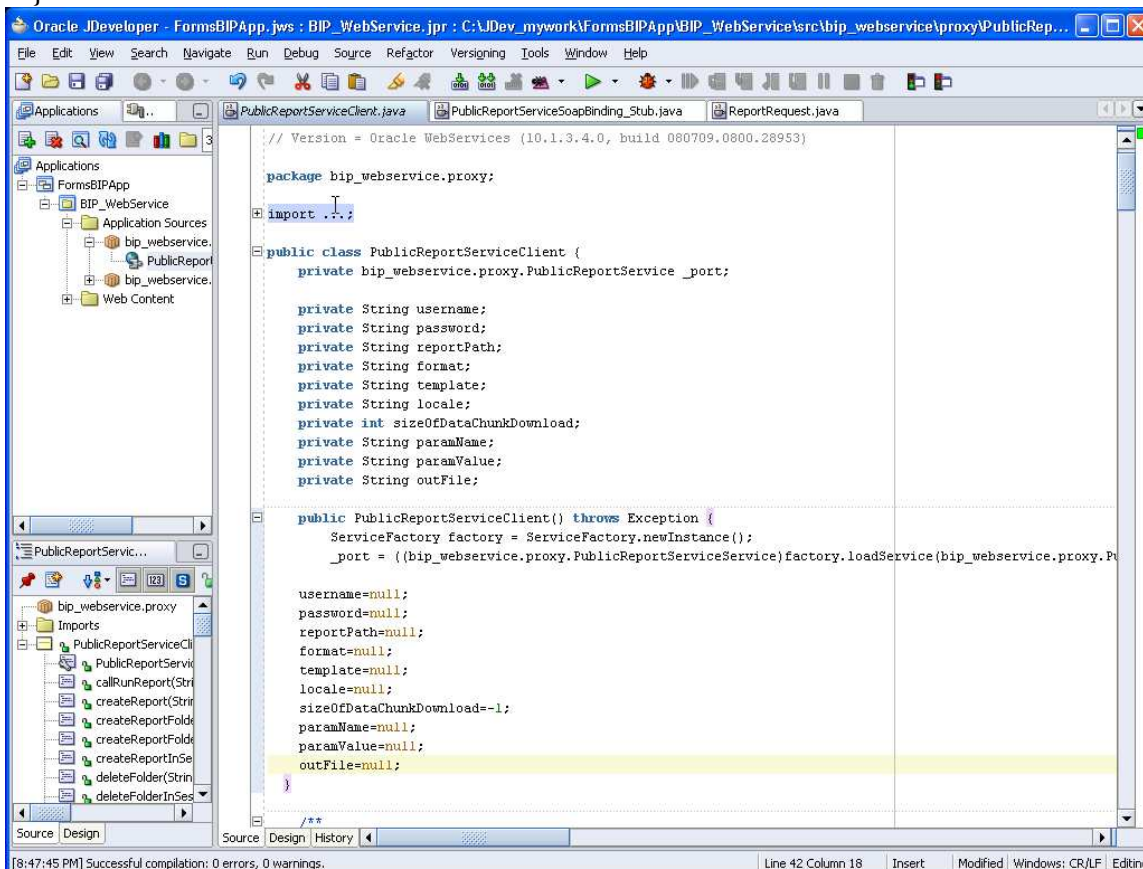
```

and initialize them

```

public PublicReportServiceClient() throws Exception {
    ServiceFactory factory = ServiceFactory.newInstance();
    _port =
        ((bip_webservice.proxy.PublicReportServiceService)factory.loadService(bip_webservice.proxy.
        PublicReportServiceService.class)).getPublicReportService();
    username=null;
    password=null;
    reportPath=null;
    format=null;
    template=null;
    locale="en-US"
    sizeOfDataChunkDownload=-1;
    paramName=null;
    paramValue=null;
    outFile=null;
}

```

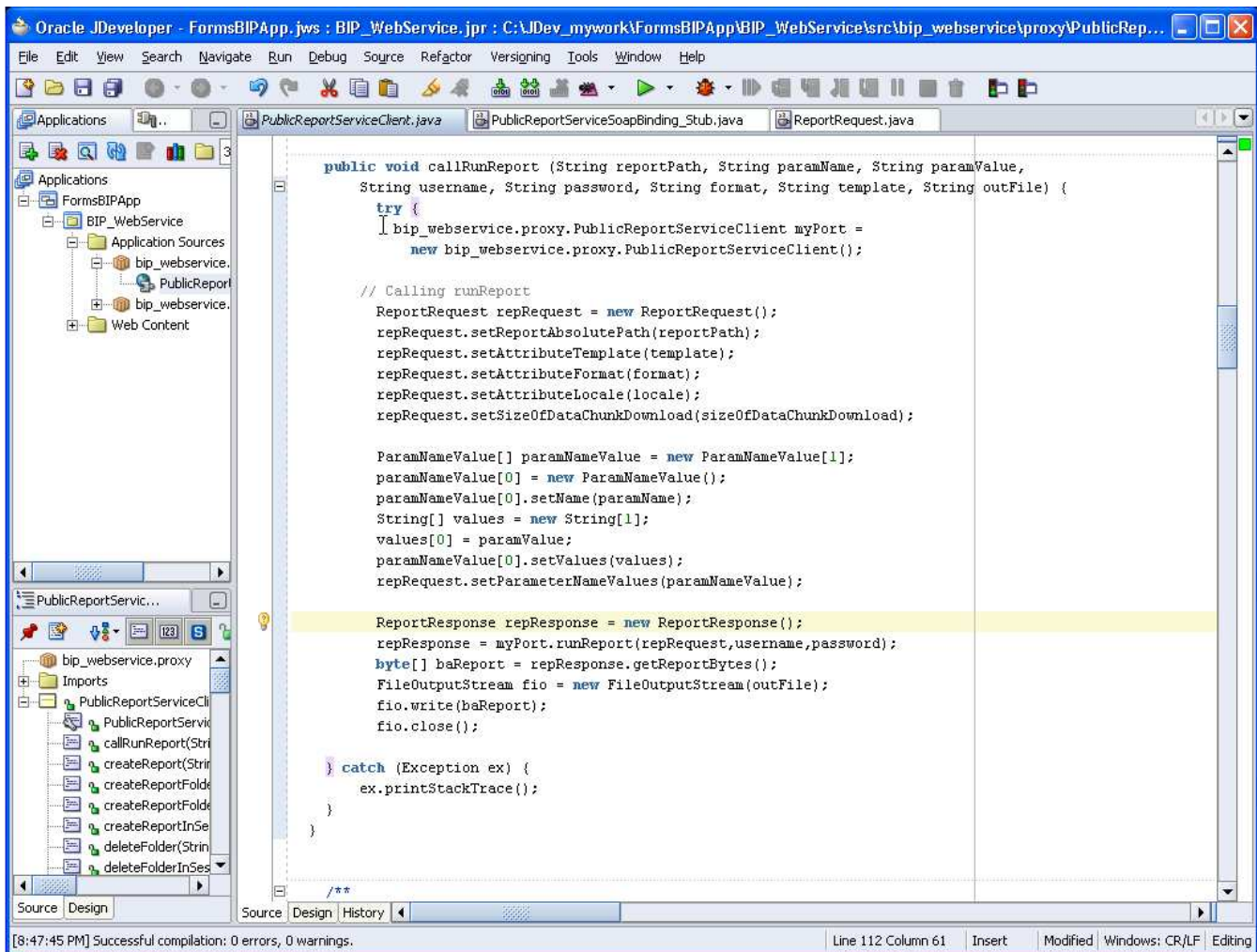


Now we will add a public method *callRunReport()*. This method will call the generated method *runReport()* in the web service client.

ParamNameValue is an object we will need to pass parameters (pairs of name and values) to the web service. You can run a report with multiple parameters and for every parameter you can have multiple values. The name and values will be passed as a nested array.

For the sake of simplicity we will reduce our example to only one parameter and only one value for it. An example how to pass an array of parameters can be found in Appendix B.

```
public void callRunReport (String reportPath, String paramName, String paramValue,  
    String username, String password, String format, String template, String outFile) {  
  
    try {  
        bip_webservice.proxy.PublicReportServiceClient myPort =  
            new bip_webservice.proxy.PublicReportServiceClient();  
  
        // Calling runReport  
        ReportRequest repRequest = new ReportRequest();  
        repRequest.setReportAbsolutePath(reportPath);  
        repRequest.setAttributeTemplate(template);  
        repRequest.setAttributeFormat(format);  
        repRequest.setAttributeLocale(locale);  
        repRequest.setSizeOfDataChunkDownload(sizeOfDataChunkDownload);  
  
        ParamNameValue[] paramNameValue = new ParamNameValue[1];  
        paramNameValue[0] = new ParamNameValue();  
        paramNameValue[0].setName(paramName);  
        String[] values = new String[1];  
        values[0] = paramValue;  
        paramNameValue[0].setValues(values);  
        repRequest.setParameterNameValues(paramNameValue);  
  
        ReportResponse repResponse = new ReportResponse();  
        repResponse = myPort.runReport(repRequest,username,password);  
        byte[] baReport = repResponse.getReportBytes();  
        FileOutputStream fio = new FileOutputStream(outFile);  
        fio.write(baReport);  
        fio.close();  
  
    } catch (Exception ex) {  
        ex.printStackTrace();  
    }  
}
```



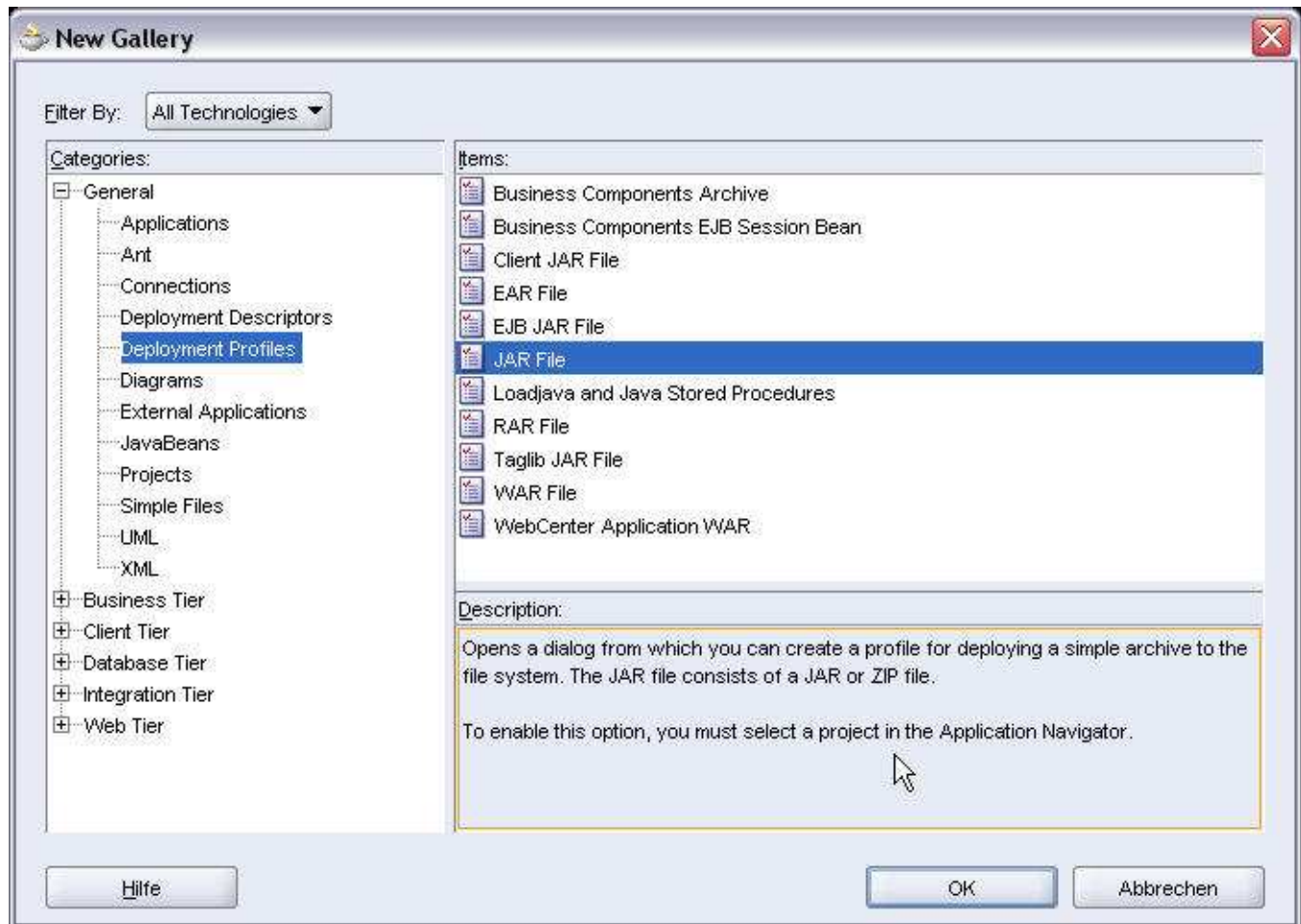
Hint:

The address of the webservice endpoint (i.e. your host, port and application) is defined in the file *PublicReportServiceSoapBinding_Stub.java*.

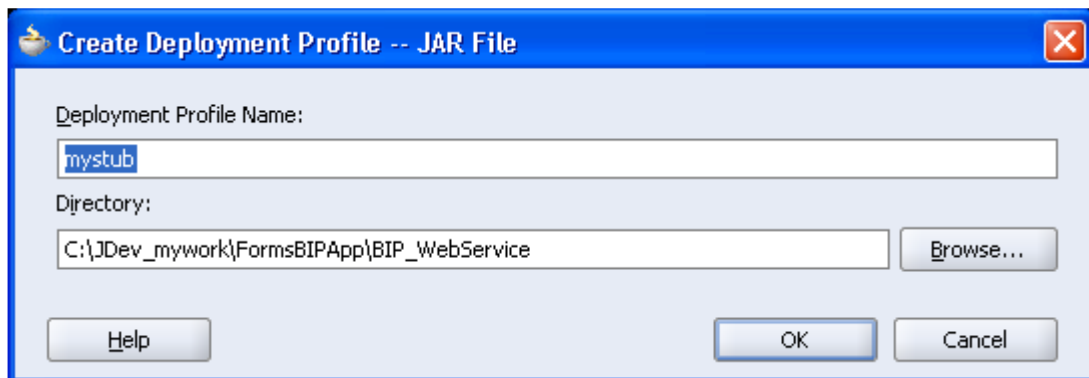


You have built and tested the Web Service Proxy from within JDeveloper. In order to call the Web Service from Forms, the proxy must be deployed to the file system as a JAR file. (As an alternative to a JAR file it's possible to use the generated files directly).

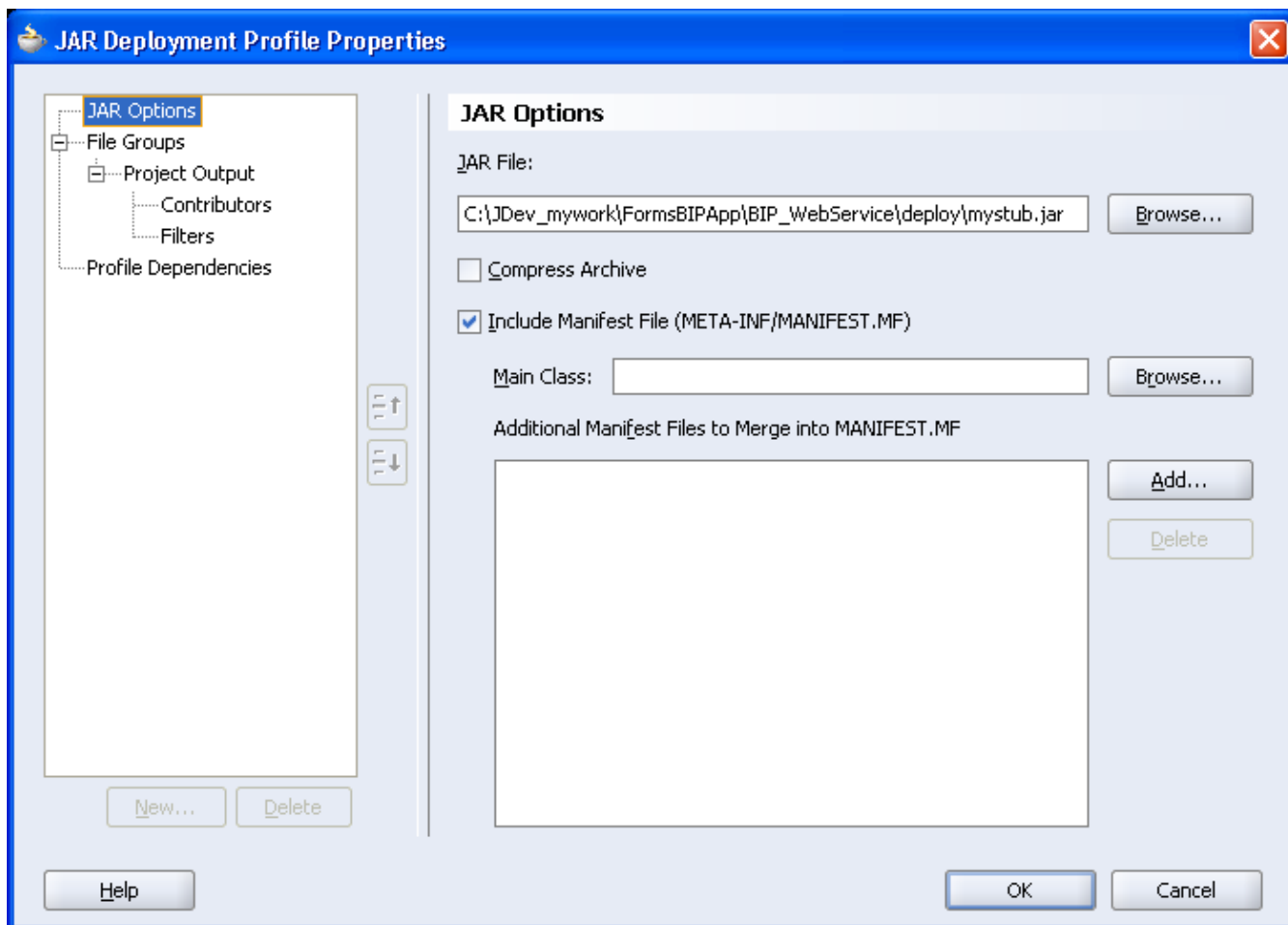
Right-click the project and select New from the context menu. There choose as item **JAR File** in the **Category: General – Deployment Profiles**.



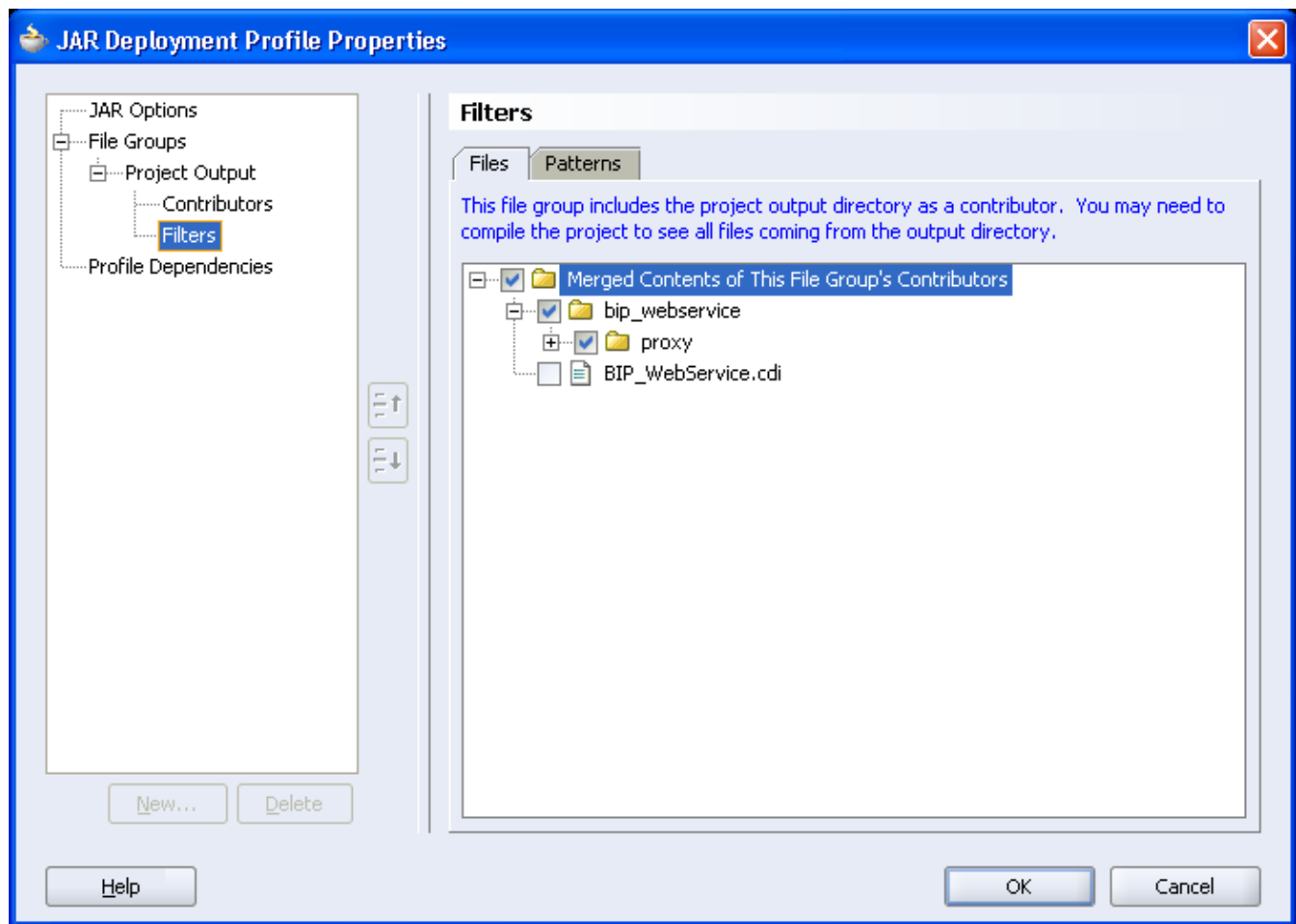
You can now choose the deployment profile name (here **mystub**)



Accept the JAR Deployment Profile Properties



Selecting the option *Filters* you can check if all the necessary files are selected.



Right-click on the archive file under the **Resources** node and select **Deploy to JAR file**. This will deploy the JAR file to the file system.



Importing and using the client in Oracle Forms

Oracle Forms needs to be able to see the relevant Java files in the Forms Builder during design time and in the OracleAS Forms Service during runtime.

Forms Builder

Java classes which have to be imported must be visible in the Classpath of the Forms Builder. You need at least the generated jar-File (*mystub.jar*) with the web service proxy and some other classes to call web services from Forms 10.1.2.x which are combined for convenience into a jar-File.

You can download and extract the jar file from:

http://download.oracle.com/otn/java/oc4j/101320/wsclient_extended_101320.zip

There are two possible ways to expose the necessary classes to Forms Builder:

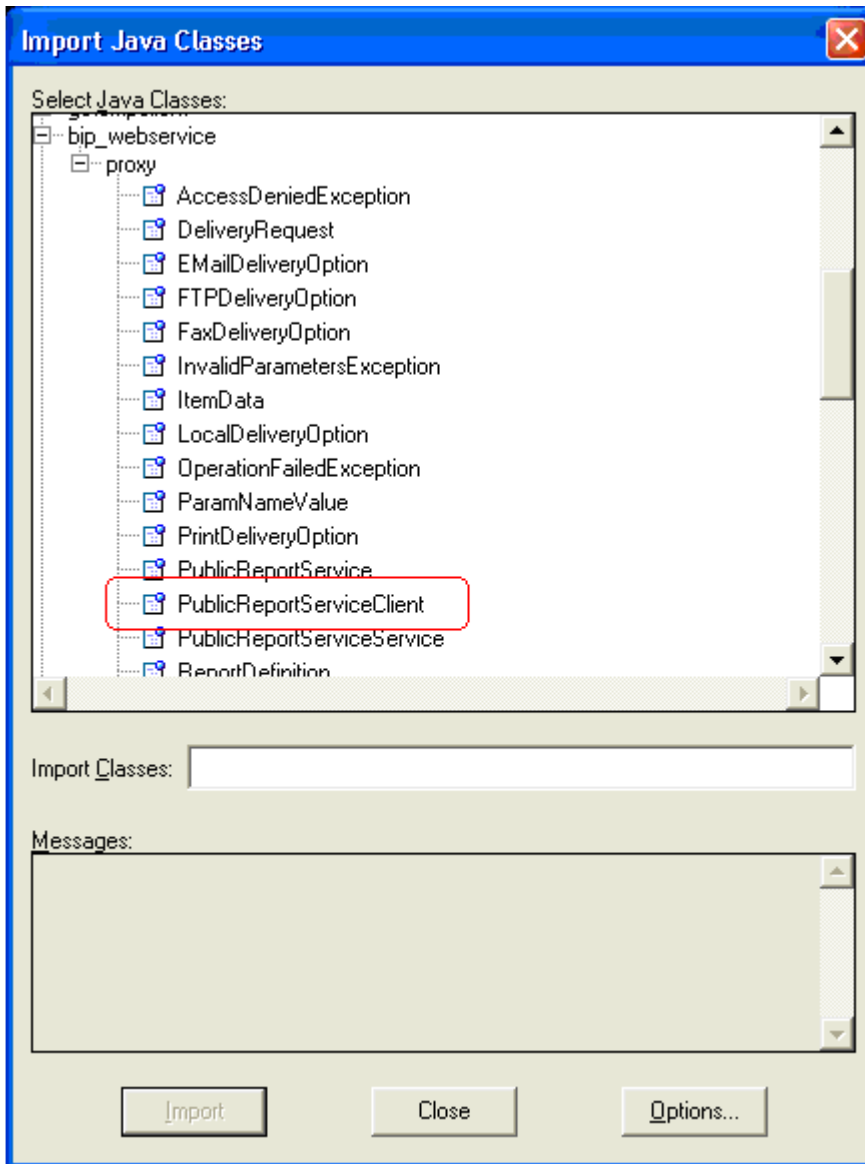
- Enter all necessary jar-Files including their absolute path in the registry key FORMS_BUILDER_CLASSPATH for the Oracle Home of your Developer Suite installation.
- Set the environment variable FORMS_BUILDER_CLASSPATH in a script and call the Forms Builder from this script.

Here is an example:

```
set
FORMS_BUILDER_CLASSPATH=C:\ora_ods_1012\jdk\jre\lib\rt.jar;C:\ora_ods_1012\forms\java\
frmbld.jar;C:\ora_ods_1012\jlib\importer.jar;C:\ora_ods_1012\jlib\debugger.jar;C:\ora_ods_1012\jli
b\utj.jar;C:\ora_ods_1012\jlib\dfc.jar;C:\ora_ods_1012\jlib\help4.jar;C:\ora_ods_1012\jlib\oracle_ic
e.jar;C:\ora_ods_1012\jlib\jewt4.jar;C:\ora_ods_1012\jlib\ewt3.jar;C:\ora_ods_1012\jlib\share.jar;C
:\ora_ods_1012\forms\java\frmwebutil.jar;C:\ora_ods_1012\forms\java\frmall.jar;C:\Web\Class;C:\
ora_ods_1012\soap\lib\soap.jar;C:\JDev10134\webservices\lib\wsclient.jar;C:\Web\Class\wsclient_
extended_101320.jar;C:\Web\Class\getempclient.jar;C:\Web\Class\mystub.jar

C:\ora_ods_1012\bin\frmbld.exe
```

In the Forms Builder create a new Forms module and import the class *bip_webservice.proxy.PublicReportServiceClient* by using the Java Importer from the menu **Program - Import Java Classes ...**

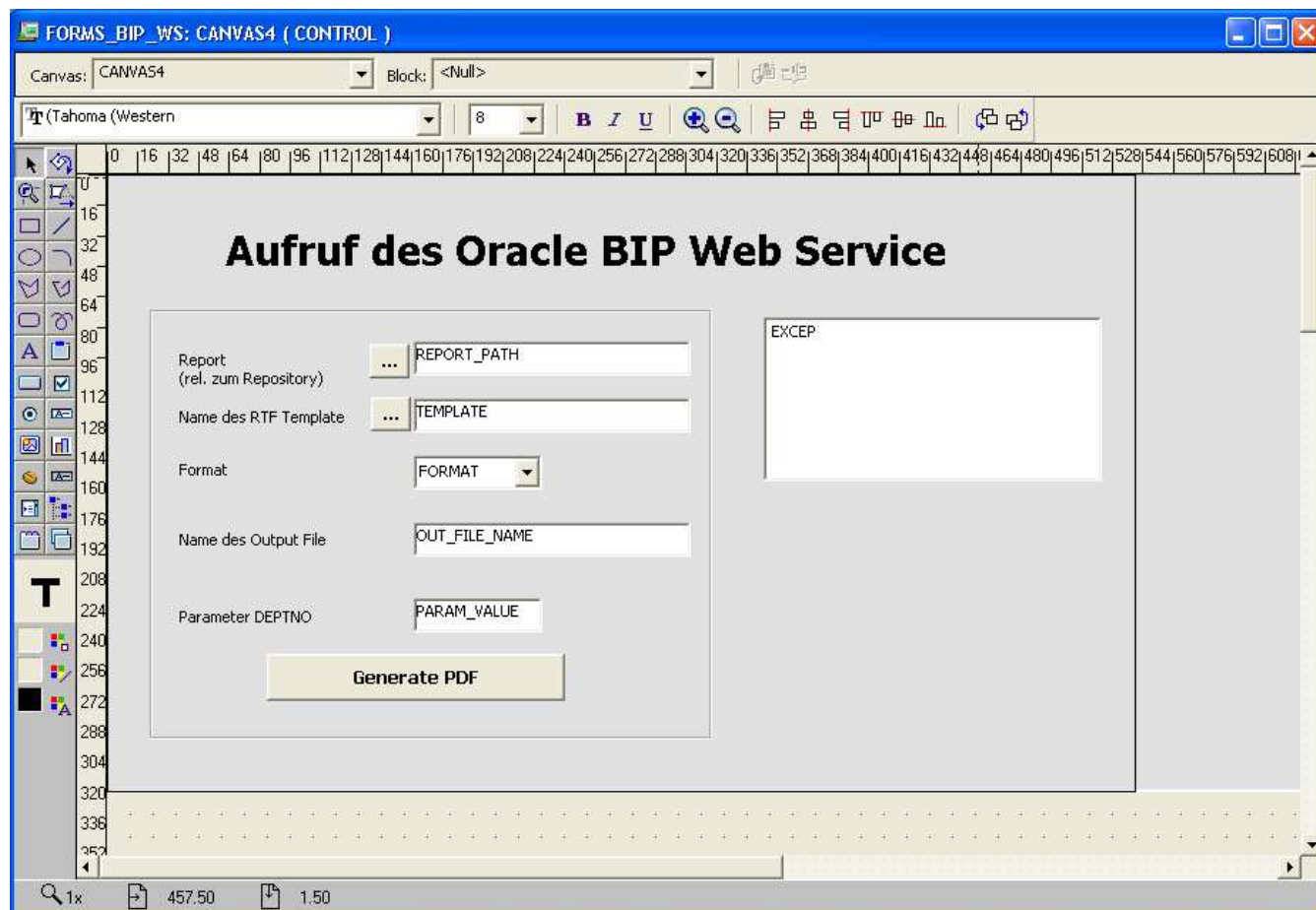


If the class is not accessible you have to modify the classpath and restart the Forms Builder.
If the class is accessible but you get error messages during the import probably some referenced classes are missing in the classpath. Add them and restart the Forms Builder.

If the import succeeds you will get a package *PUBLICREPORTSERVICECLIENT*.

Now you can define your application logic in Forms and call the BI Publisher web service by using the PL/SQL procedure *PublicReportServiceClient.callRunReport()*.

Here is an example how the code could like like. The code assumes that there's a block *CONTROL* with the items *REPORT_PATH*, *FORMAT*, *TEMPLATE*, *OUT_FILE_NAME*, *PARAM_VALUE* and a button *GENERATE_PDF*.



Here is the program code of the trigger WHEN-BUTTON-PRESSED:

DECLARE

```

param_name varchar2(200);
un varchar2(200);
pw varchar2(200);
out_file varchar2(200);
obj ORA_JAVA.OBJECT;

```

BEGIN

```

obj      := publicreportserviceclient.new();
param_name := 'p_deptno';
un       := 'Administrator';
pw       := 'Administrator';
out_file  := 'C:\ora_ods_1012\j2ee\home\default-web-app\':out_file_name;

```

```

publicreportserviceclient.callRunReport(obj,:report_path,param_name,:param_value,un,
pw,:format,:template,out_file);
web.show_document('http://jmenge-de.de.oracle.com:8889/j2ee/' || :out_file_name);

```

END;

The report will be created as a file on the application server. Afterwards this file will be displayed with the built-in `web.show_document()`. For a multi-user environment you need to generate unique filenames. The file in this simple example is generated into `c:\ora_ods_1012\j2ee\home\default-web-app` which is a virtual directory for the local OC4J in the Oracle Developer Suite. The sample code has no error handling (which would be a good idea to have).

Parameters

report_path:	the location of the report definition file relative to the BI Publisher repository (example: <code>/HR Manager/Employee Salary Report/Employee Salary Report.xdo</code>).
param_name:	The name of the parameter which is passed to the BI Publisher web service.
param_value:	The value of the parameter which is passed to the BI Publisher web service.
un / pw:	Username and Password for authentication against the BI Publisher server
format:	The desired format of the document. Valid values are pdf, rtf, excel, xml and html. If in the BI Publisher Environment a specific format is not allowed, it's not generated via web services too.
template:	The name of the template (take the logical name of the template, not the physical one).
out_file:	Directory path and name of the file which has to be generated.

Synchronous vs. asynchronous call

The example uses a synchronous call of the BI Publisher Web Service, so you can call the document via `web.show_document` directly after the call of the method `callRunReport()`. For long running reports it would be better, to call the reports asynchronous, which could be done via a multi-threaded web service client.

OracleAS Forms Service

In the runtime environment you again need all the used and referenced classes in the classpath. This time the classpath has to be defined in the environment file of the Forms Service (*default.env*). It should contain a list of required jar-files including their absolute path on the application server machine. Here is an example of the CLASSPATH in *default.env*:

```
CLASSPATH=C:\ora_ods_1012\jdk\jre\lib\rt.jar;  
C:\ora_ods_1012\j2ee\OC4J_BI_Forms\applications\formsapp\formsweb\WEB-  
INF\lib\frmsrv.jar; C:\ora_ods_1012\jlib\repository.jar;C:\ora_ods_1012\jlib\ldapjclnt10.jar;  
C:\ora_ods_1012\jlib\debugger.jar;C:\ora_ods_1012\jlib\ewt3.jar;  
C:\ora_ods_1012\jlib\share.jar;C:\ora_ods_1012\jlib\utj.jar;C:\ora_ods_1012\jlib\zrclient.jar;  
C:\ora_ods_1012\reports\jlib\rwrun.jar;C:\ora_ods_1012\forms\java\frmwebutil.jar;  
C:\Web\Class;C:\Web\Class\wsclient_extended_101320;  
C:\Web\Class\mystub.jar;C:\ora_ods_1012\soap\lib\soap.jar;  
C:\JDev10134\webservices\lib\wsclient.jar;C:\JDev10134\webservices\lib\jaxrpc-api.jar;  
C:\JDev10134\webservices\lib\saaj-api.jar;C:\JDev10134\webservices\lib\orasaaj.jar;  
C:\JDev10134\webservices\lib\xsdlib.jar;C:\JDev_mywork\XMLP\jlib\xdocore.jar;  
C:\JDev_mywork\XMLP\jlib\xdoparser.jar;C:\JDev_mywork\XMLP\jlib\xmlparserv2-904.jar;  
C:\JDev_mywork\XMLP\jlib\collections.jar;C:\JDev_mywork\XMLP\jlib\versioninfo.jar;  
C:\JDev_mywork\XMLP\jlib\i18nAPI_v3.jar;C:\JDev10132\jdbc\lib\ojdbc14dms.jar;  
C:\JDev10134\jdbc\lib\ojdbc14.jar;C:\JDev10134\jdbc\lib\orai18n.jar;  
C:\JDev_mywork\XMLP\jlib\jewt4.jar;C:\JDev10134\jdbc\lib\ocrs12.jar;  
C:\JDev10134\diagnostics\lib\ojdl.jar;C:\JDev10134\lib\dms.jar;  
C:\JDev10134\j2ee\home\oc4jclient.jar
```

It is recommended to have a separate configuration in the *formsweb.cfg* and to define here a specific environment file for this application.

```
[bip_webservice]  
envFile=bip_webservice.env  
...
```

Test the form by running the named configuration and call the BI Publisher report from your application:

```
http://<host>:<port>/forms/frmservlet?config=bip_webservice& ...
```

Debugging

When running the Form for the first time you perhaps will get an unhandled exception with ORA-105100 or ORA-105101. This indicates that Java classes are still missing in your runtime environment or that a Java error occurred during the execution of your code.

It would be helpful to have some more information what happened by using some kind of exception handling. There is a way to display the Java error stack in your Form which could be of great help finding the cause of the error.

Duncan Mills and Jan Carlin (from Oracle) have described a solution in their blogs:

[http://groundside.com/blog/DuncanMills.php?title=exception handling in forms java integra&more=1&c=1&tb=1&pb=1](http://groundside.com/blog/DuncanMills.php?title=exception%20handling%20in%20forms%20java%20integra&more=1&c=1&tb=1&pb=1)

[http://groundside.com/blog/JanCarlin.php?title=extracting a stack trace from a java exc&more=1&c=1&tb=1&pb=1](http://groundside.com/blog/JanCarlin.php?title=extracting%20a%20stack%20trace%20from%20a%20java%20exc&more=1&c=1&tb=1&pb=1)

If you still get unhandled exceptions during the execution of your Form you probably have to modify the code from both examples. In the code example below a nested exception catches the errors during the execution of the *Exception_.toString()*.

The modified parts are marked.

```
exception
--check for ORA-105100
  when ORA_JAVA.JAVA_ERROR then
    message('Unable to call out to Java, ' ||ORA_JAVA.LAST_ERROR);
    return;
--check for ORA-105101
  when ORA_JAVA.EXCEPTION_THROWN then
    raisedException := exception_.new(ORA_JAVA.LAST_EXCEPTION);
    begin
      :control.excep := 'Exception: '||Exception_.toString(raisedException);
      exception
        when ORA_JAVA.JAVA_ERROR then
          message('Unable to call out to Java, ' ||ORA_JAVA.LAST_ERROR);
          return;
        end;
--Get an array of StackTraceElement from the Exception
  stack_trace:=Exception_.getStackTrace(raisedException);
--Loop over all the Elements
  for i in 0..ora_java.get_array_length(stack_trace) loop
--Get each Element
  stackTrcElement:=ora_java.get_object_array_element(stack_trace, i);
--Make a string out of it and add it to the error field
  :control.excep:=:control.excep||(10)||stackTraceElement.toString(stackTrcElement);
  end loop;
  ORA_JAVA.CLEAR_EXCEPTION;
  return;
  when OTHERS then
    message('Problem!');
    return;
end;
```

Outlook

The example could be extended in many ways:

- It is possible to pass also complex parameter structures with more than one parameter and more than one value per parameter (multiple selection) by using the class *ParamNameValue* from the web service proxy.

In that case you have to import the class *bip_webservice.proxy.ParamNameValue* into your form. First you have to create an instance of that class in Forms with:

```
a0 := paramnamevalue.new();
```

Then you can use the methods *setName()* and *setValues()* to provide the necessary parameters to the object. Finally you call the web service and pass this object as a parameter.

- Another direction could be to separate the processing of the report from the Forms application by using an asynchronous call of BI Publisher Web Service (see above).
- It is also possible to write the generated report as a byte stream into a CLOB column in the database which would give us the opportunity to use features from the database (security, stored procedures, AQ etc.).

Disclaimer

This document is not an official Oracle product documentation. Therefore following the described approach is at your own risk and there is no claim for support from Oracle Support Service. If you need help we recommend to engage Oracle Consulting.

Contact

Dr. Jürgen Menge
Principal Sales Consultant
juergen.menge@oracle.com

Rainer Willems
Principal Sales Consultant
rainer.willems@oracle.com

Appendix A Report *Employees.xdo*

```
<?xml version = '1.0' encoding = 'utf-8'?>
<report version="1.1" xmlns="http://xmlns.oracle.com/oxp/xmlp"
defaultDataSourceRef="demo">
  <title>Employees</title>
  <description>Used for Web Service demonstration</description>
  <properties>
    <property name="showControls" value="true"/>
    <property name="online" value="true"/>
    <property name="parameterColumns" value="3"/>
    <property name="openLinkInNewWindow" value="true"/>
    <property name="autoRun" value="true"/>
  </properties>
  <dataModel defaultDataSet="Employees">
    <dataSet id="Employees">
      <sql dataSourceRef="scott">
        <![CDATA[select      DEPT.DNAME as DNAME,
        EMP.DEPTNO as DEPTNO,
        EMP.SAL as SAL,
        EMP.JOB as JOB,
        EMP.ENAME as ENAME,
        EMP.EMPNO as EMPNO,
        DEPT.LOC as LOC
from      SCOTT.EMP EMP,
        SCOTT.DEPT DEPT
where     EMP.DEPTNO=DEPT.DEPTNO
and      emp.deptno=nvl(:p_deptno,emp.deptno)
and      emp.job=nvl(:p_job,emp.job)]]>
      </sql>
      <input id="p_deptno" value="{p_deptno}" dataType="xsd:integer"/>
      <input id="p_job" value="{p_job}" dataType="xsd:string"/>
    </dataSet>
  </dataModel>
  <valueSets>
    <valueSet id="LOV_DEPTNO">
      <sql dataSourceRef="scott">
        <![CDATA[select dname, deptno from dept]]>
      </sql>
    </valueSet>
    <valueSet id="LOV_JOB">
      <sql dataSourceRef="scott">
        <![CDATA[select      EMP.JOB as JOB from
        SCOTT.EMP EMP]]>
      </sql>
    </valueSet>
  </valueSets>
  <parameters>
    <parameter id="p_deptno" dataType="xsd:integer">
      <select label="Deptno" valueSet="LOV_DEPTNO" multiple="false" all="true"
allValue="null"/>

```

```
</parameter>
<parameter id="p_job">
  <select label="Job" valueSet="LOV_JOB" multiple="false" all="true" allValue="null"/>
</parameter>
</parameters>
<templates>
  <template label="Simple" type="rtf" url="Simple.rtf"/>
</templates>
<burst enabled="false"/>
</report>
```

Appendix B Method callRunReport() with an array of parameters

```
public void callRunReport (String reportPath, String[] paramName, String[] paramValue, String
username, String password, String format, String template, String outFile)
{
    try {
        bip_webservice.proxy.PublicReportServiceClient myPort =
        new bip_webservice.proxy.PublicReportServiceClient();
        // Calling runReport
        ReportRequest repRequest = new ReportRequest();
        repRequest.setReportAbsolutePath(reportPath);
        repRequest.setAttributeTemplate(template);
        repRequest.setAttributeFormat(format);
        repRequest.setAttributeLocale("en-US");
        repRequest.setSizeOfDataChunkDownload(-1);

        if (paramName != null)
        {
            ParamNameValue[] paramNameValue = new ParamNameValue[paramName.length];
            String[] values = null;
            for (int i=0; i<paramName.length; i++)
            {
                paramNameValue[i] = new ParamNameValue();
                paramNameValue[i].setName(paramName[i]);
                values = new String[1];
                values[0] = paramValue[i];
                paramNameValue[i].setValues(values);
            }
            repRequest.setParameterNameValues(paramNameValue);
        }
        else
            repRequest.setParameterNameValues(null);

        ReportResponse repResponse = new ReportResponse();
        repResponse = myPort.runReport(repRequest,username,password);
        byte[] baReport = repResponse.getReportBytes();
        FileOutputStream fio = new FileOutputStream(outFile);
        fio.write(baReport);
        fio.close();

    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```