ORACLE®
FUSION MIDDLEWARE
APPLICATION DEVELOPMENT
FRAMEWORK

Integrating Oracle BI Publisher Reports into Oracle ADF Applications

ORACLE®

Table of Contents

Purpose of this Paper

This paper describes the integration of enterprise reporting using Oracle BI Publisher into custom applications build with Oracle's Application Development Framework (ADF). The target audiences are mainly developers with basic knowledge of Oracle BI Publisher, Oracle ADF and Java. The example consists of two ADF applications which can be downloaded. The base application needs only small modifications to fit into the reader's technical environment. The consuming application only contains an empty page and has to be completed by the reader. The necessary steps are described in the following paper.

Introduction to Oracle ADF and Oracle BI Publisher

Built on top of Java Enterprise Edition (Java EE), Oracle Application Development Framework (ADF) provides a complete and integrated framework for enterprise web application and mobile browser application development. Within Oracle, ADF is used as the foundation of Oracle Fusion Applications, Oracle Cloud and On-premise Fusion Middleware products. Outside of Oracle, a constantly growing number of customers use Oracle ADF for building custom software solutions, or to extend the functionality of Fusion Applications.
For more information about Oracle ADF see:
http://www.oracle.com/technetwork/developer-tools/adf/overview.

With Oracle ADF Faces, the JavaServer Faces based view layer framework in ADF, developers build web pages declaratively and component based, including pages that are for information purposes like reporting.
Often the requirement in enterprise reporting is to produce high-quality reports that use pixel-perfect layouts and that are printed in document formats like PDF, MS Word, MS Excel or MS PowerPoint. This is where another Oracle product, Oracle BI Publisher, can be used to complement Oracle ADF online reporting capabilities.
Oracle BI Publisher is a modern, standard-based solution to develop, generate and distribute various types of business reports and documents. It is available to customers along with BI Suite of products and can also be installed as a standalone product. BI Publisher is also used by over 80 Oracle Products such as Enterprise Manager, Siebel CRM, and Oracle Financial Services etc. For more information about Oracle BI Publisher see:
http://www.oracle.com/technetwork/middleware/bi-publisher/overview.

Oracle BI Publisher provides three options for integrating with Oracle ADF:

» **Executing a report by calling a URL**
» **Using the BI Publisher Java API**
» **Using the BI Publisher web service API**

The BI Publisher web service API is powerful and comprehensive. Oracle recommends this integration approach and therefore this whitepaper will focus on this approach.

For more information about Oracle BI Publisher web service API see:
http://docs.oracle.com/cd/E28280_01/bi.1111/e22259/toc.htm.

Getting Started

You can download the sample code for this whitepaper. To work through the steps and complete the example you need the following software installed on your computer.

Oracle BI Publisher

In this example version 11.1.1.7.1 of BI Publisher is used. But it should also work with previous versions of the 11g release without modifications. For the ADF integration, this whitepaper however relies on the BI Publisher V2 web services introduced in release 11*g*.

BI Publisher 11*g* can be used as part of the Oracle BI EE installation or from the Trial Edition because the required web services are available in both installations.

In BI Publisher's server configuration the security model is set to „BI Publisher Security" but it should work also with other security models.
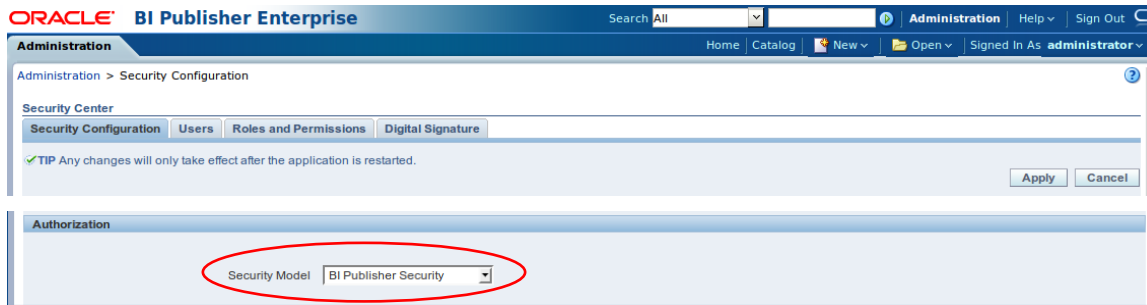

Figure 1 - Configuration of security model in Oracle BI Publisher

The example uses a set of sample reports provided by Oracle. So there is no need to create your own reports. Only the data sources of the sample reports have to be configured in a way that the required data can be found.

Oracle ADF

The ADF applications of this example are developed with Oracle JDeveloper 12c (12.1.3.) but also previous versions of JDeveloper or Oracle Enterprise Pack for Eclipse (OEPE) could be used for this purpose. Because the provided sample application was produced with JDeveloper 12.1.3 it is not possible to import it in previous versions of JDeveloper or in OEPE. In this case the sample applications could be used as a blueprint including custom Java code which is reusable.

Implementation

The example consists of two JDeveloper applications

» **Base application *BipClientApplication***
contains the required functionality to generate reports using the web service API of BI Publisher.
This application will be provided with the downloaded sample code and needs only some small modifications
(host, port, user, password, etc. of BI Publisher server).

» **Consuming application *BIPublisherApp*** where the functionality from the base application will be integrated.
This application has to be developed from scratch based on the description in this paper.

Base Application

What was already done for you

The base application *BipClientApplication* includes two projects *BipWebService* and *BipInterface* and is provided as a sample.

The project *BipWebService* consists of the following parts:

» **web service client** communicating with BI Publisher server
» **wrapper code to** prepare the required data structures and methods
» **data control** to expose the required data structures and methods

The project *BipInterface* consists of the following parts:

» **bounded task** flow to control the process of selecting report parameters and execute the report
» **page fragment** with the user interface to select the report including runtime parameters
» **test page** to display the bounded task flow as a static region in the UI

The base application was developed in the following steps. The description can only give an overview and assumes some experience with Oracle JDeveloper and ADF.

(1)   First a JAX-WS client has to be gnerated for the BI Publisher web services

In this step a running instance of BI Publisher server 11.1.1.7.x is required (see Getting Started => Oracle BI Publisher). The client was generated using the wizard *Create Web Service Client and Proxy* into package *oracle.bip.webservice*.
From the four existing web service endpoints of BI Publisher 11.1.1.7.x we used the following three in our example:

> *http://<host>:<port>/xmlpserver/services/v2/ReportService?wsdl*
> *http://<host>:<port>/xmlpserver/services/v2/SecurityService?wsdl*
> *http://<host>:<port>/xmlpserver/services/v2/CatalogService?wsdl*

(2)   In order to generate the data controls based on Java classes (POJO) we need some wrapper code that exposes the required data structures and methods to the view layer.
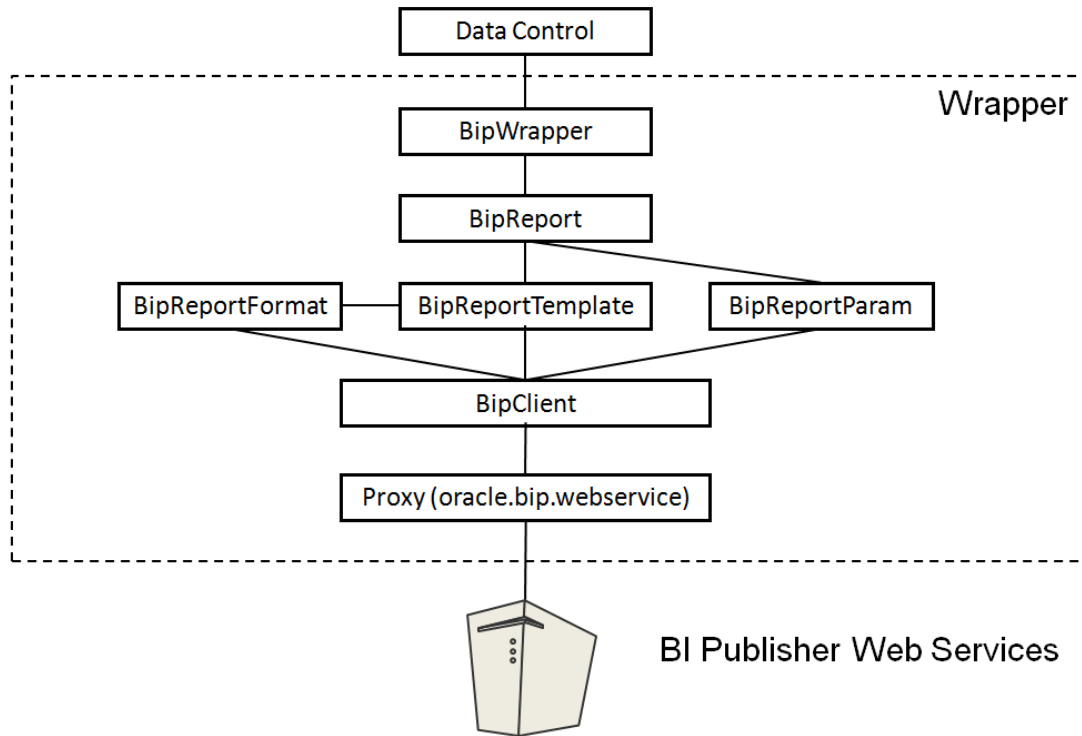Here is a picture of the class structure the wrapper contains.

Figure 2 - Java classes of web service wrapper

These classes are grouped in package *oracle.bip.client.ui*.

The method *main()* in *BipWrapper* is used only for testing purposes in JDeveloper. If such a test is required the parameters (admin user, admin password, report user, repository folder) should be set to valid values in the code (see below).

Requesting the execution of a report by BI Publisher server normally requires an authentication against the security provider (in our case BI Publisher server itself). The example uses two features of BI Publisher for authentication:

» login at BI Publisher server returns a **token** (*_sessionId*) which is subsequently used in all web service calls (*inSession* methods)

» to allow the login of arbitrary users the method ***impersonate()*** is used for the login. In this case the authentication is done always for the admin user on behalf of the user who wants to execute a report. The method logs in using admin account privileges, and then switch the owner of the BI Publisher server session to the passed-in username.

(3)	When the wrapper code delivers the expected results the POJO data control can be generated for the class *BipWrapper*. This is done simply by running the option *Create Data Control* from the context menu for the class *BipWrapper*. Data control can be regenerated every time after modifications in the wrapper code (caution: to regenerate the data control the name of the data control must be identical).

(4)	Using the generated data control the user interface will be build in project *BipInterface*. The task flow *bip-taskflow.xml* consists of only two activities

» a method call *login*
» a page fragment *bip.jsff* with UI components.
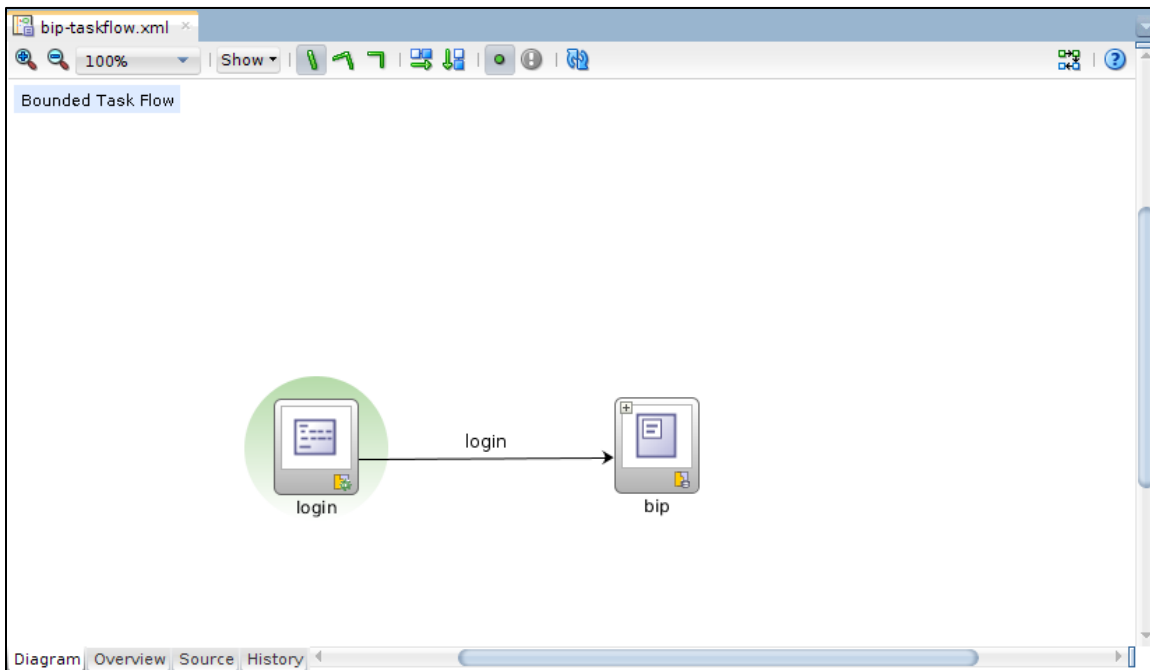
Figure 3 - BI Publisher task flow

To get the desired results the task flow has four mandatory parameters:

Input Parameter	Description	Example	Data Type
BipAdminUserName	BIP Admin user name	administrator	String
BipAdminPassword	BIP Admin password	weblogic1	String
BipUser	BIP user to impersonate. This is usually a normal user that exists in BIP. For example, each ADF application user has an account in BIP, and in order for each user to view the reports he has privilege to view, the logged in ADF user can be passed.	This could be static: appuser Or dynamic: #{SecurityContext.userName}	String
BipFolderPath	Reports path as specified in BI Publisher which resides under *Shared Folders*	*/Samples/Human Resources* Note: make sure path starts with /	String

(5)   Because a page fragment cannot be executed on its own a test *page test.jsf* is provided as part of the base application. The task flow is implemented as a static region inside the test page. When the task flow is dragged into the page values for the parameters have to be provided.

The parameters and the generated report are processed in the Java bean *oracle.bip.ui.jsf.beans.BipBean.java* which is registered in *bip-taskflow* as a managed bean with page flow scope.

What still has to be done

(1)   In the downloaded sample application the endpoint URLs have to be changed to an existing instance of BI Publisher. The best way is to use the search functionality in JDeveloper (*Search => Find in Files*) to find all occurrences of **xmlpserver** and replace host name and port number to valid values.

(2)   If the page *test.jsf* is used for testing the parameter values have to be changed to the existing BI Publisher environment by editing the task flow binding for *taskFlow - biptaskflow1*. You have to provide valid values for (see page 8):

» name and password of BIP admin user

» name of BIP user

» name of shared BI Publisher folder
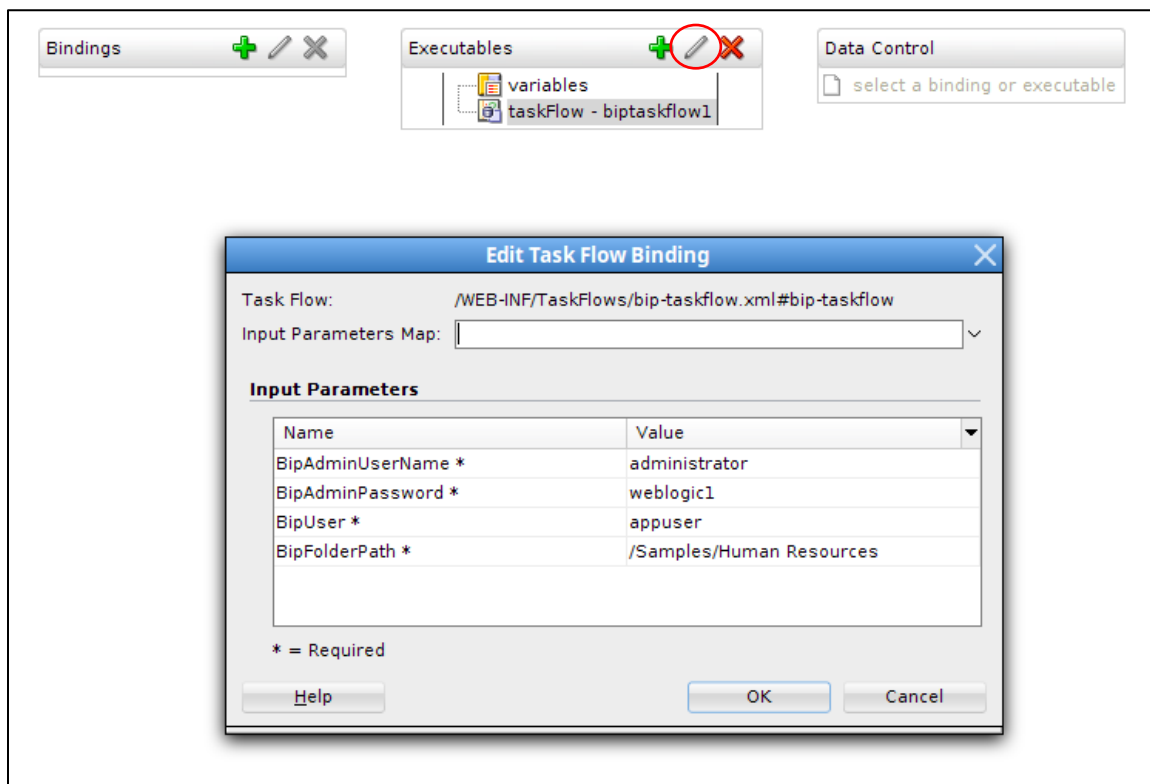
Figure 4 - - Setting parameter values in task flow binding

You can test the page *test.jsf* simply by running it using the Integrated WebLogic server of JDeveloper.

(3)　　　To consume the developed functionality in other applications is has to be exported as an ADF library jar file.

This should be done by executing the context menu option *Deploy => BipClient-taskflow* for the project *BipInterface*.The generated jar file can be found in the directory *./BipClientApplication/BipInterface/deploy* and should be copied afterwards into the directory *./Resources*.
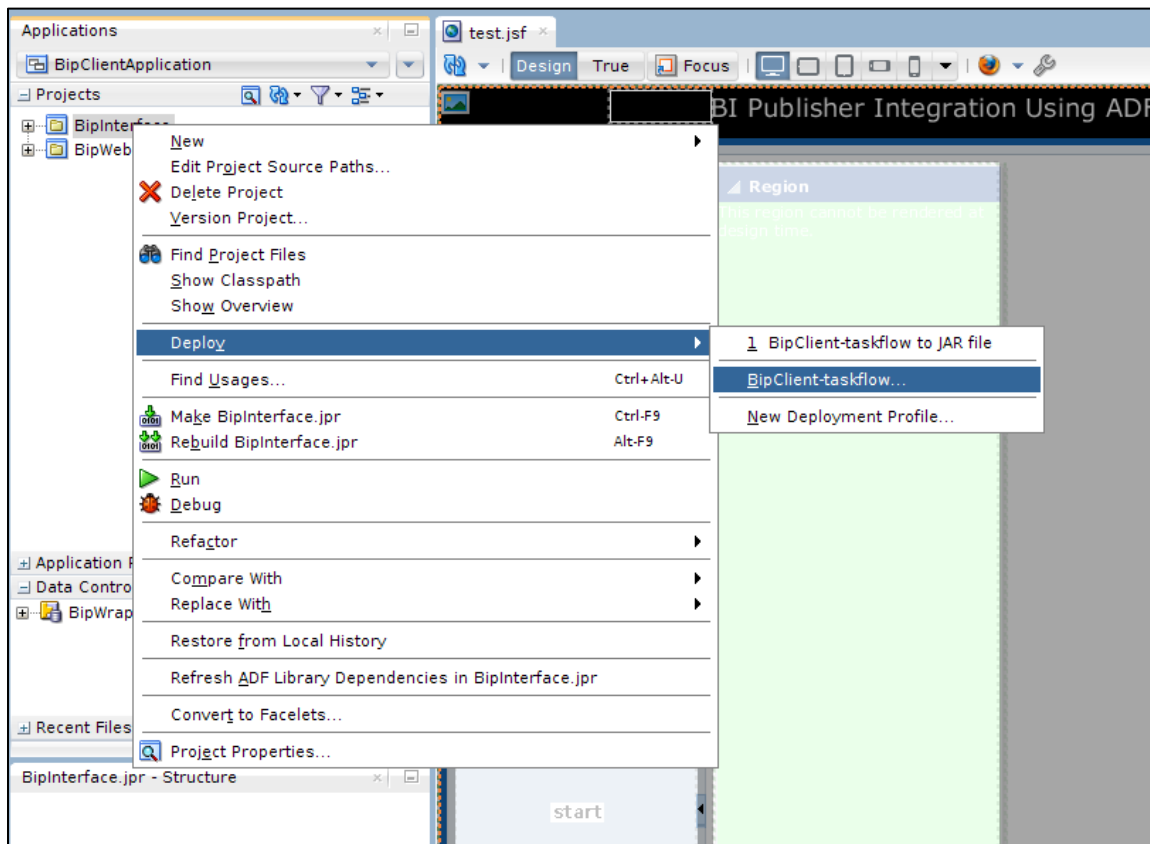
Figure 5 - Exporting BI Publisher task flow

Consuming Application

What was already done for you

The application *BIPublisherApp* contains a single page *runBIP.jsf* based on the three-column template from Oracle. The page is prepared to integrate the BI Publisher task flow produced in the base application from previous step.

What still has to be done

(1)   In this step we will add the library with the ADF task flow to the *ViewController* project of the application. Because the library is a Java Archive file (jar) we first need to create a file system connection in JDeveloper.

The library *BipClient-taskflow.jar* contains all the necessary functionality to call existing BI Publisher reports using BI Publisher web services to

» Login the specified user
» Querying the existing reports for that user
» Providing list of values for the available formats and parameters for that report
» Executing the report.

(1.1)   First it is necessary to add the BI Publisher library to the *ViewController* project of the consuming application.
Open the application *BIPublisherApp* in JDeveloper.
If not already visible open the Window *Resources* by selecting *Window => Resources* from the top menu of JDeveloper.
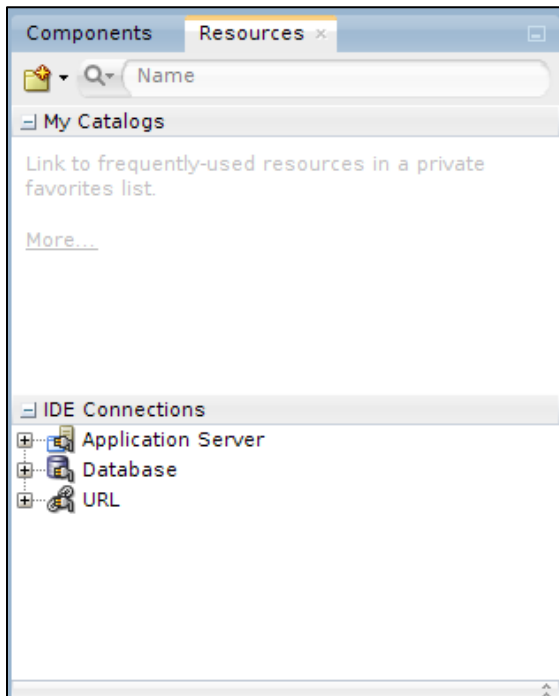
Figure 6 - Window Resources

(1.2)   Create a new File System Connection by open the New Gallery with *File => New => From Gallery* from the top menu of JDeveloper.

Select *General => Connections* at the left side of the New Gallery dialogue.

Select *File System Connection* at the right side of the New Gallery dialogue.
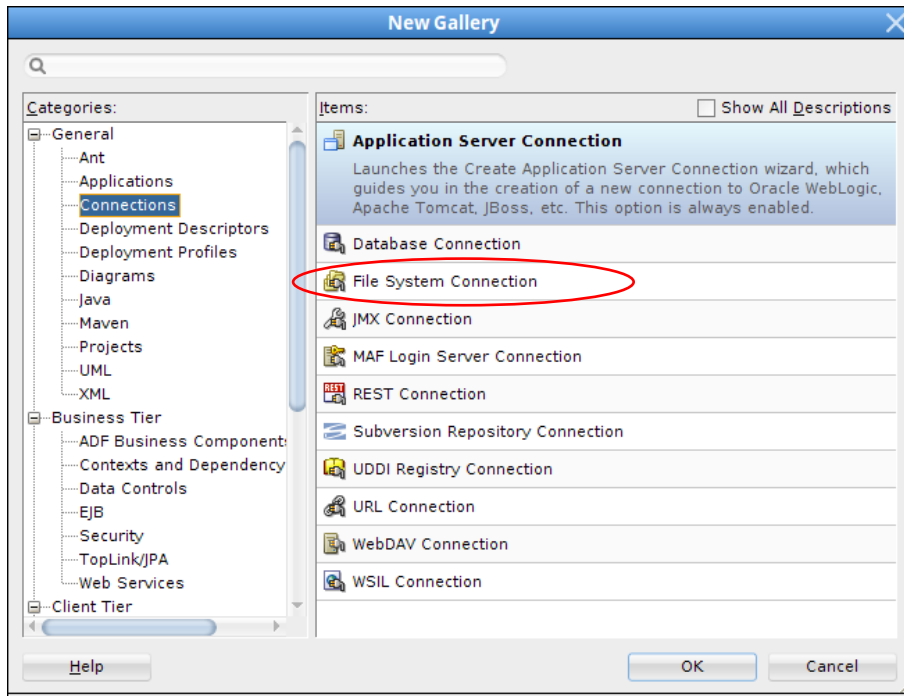
Figure 7 - Create file system connection

Enter the necessary properties for the connection.

Property	Value
Connection Name	BIPublisherDir
Directory Path	<your_working_dir>\Resources

Test the connection and accept these entries by clicking *OK*.
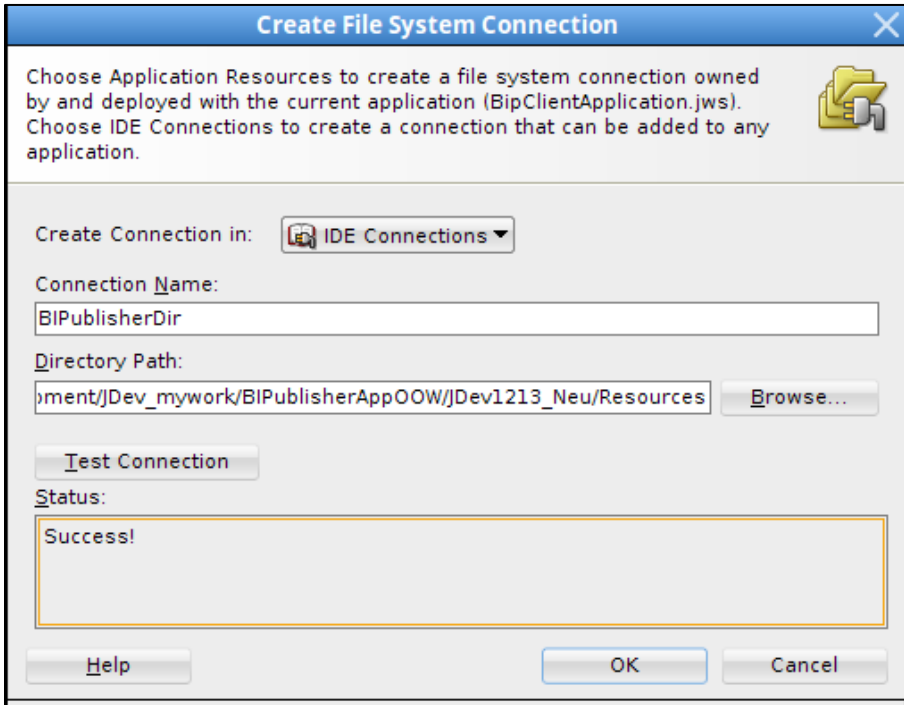
Figure 8 - Enter file system connection details

Opening the new created file system connection *BIPublisherFolder* a file *BipClient-taskflow.jar* is visible. This jar file contains an ADF task flow *bip-taskflow* which was developed in the base application and is provided as a reusable asset.
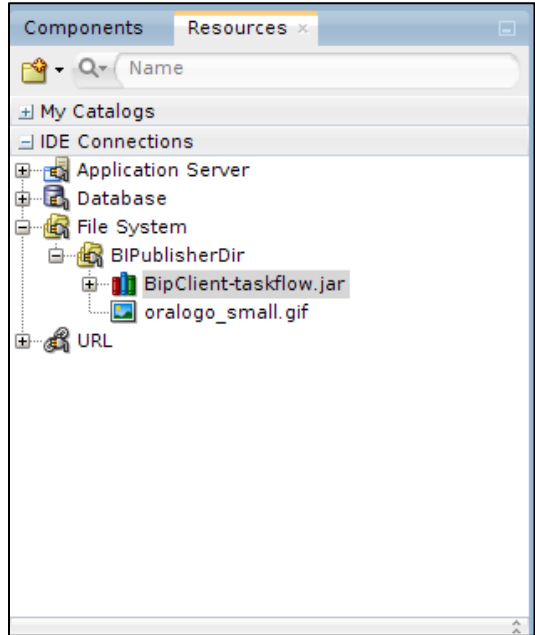
Figure 9 - Task flow in ADF library

(1.3)   Add the library file *BipClient-taskflow.jar* to the *ViewController* project:

Select the *ViewController* project in the Application Navigator

Select the library jar file *BipClient-taskflow.jar* in the window *Resources* and execute the option *Add to Project* from the context menu.

Confirm the alert that you want to add the library to the *ViewController* project.
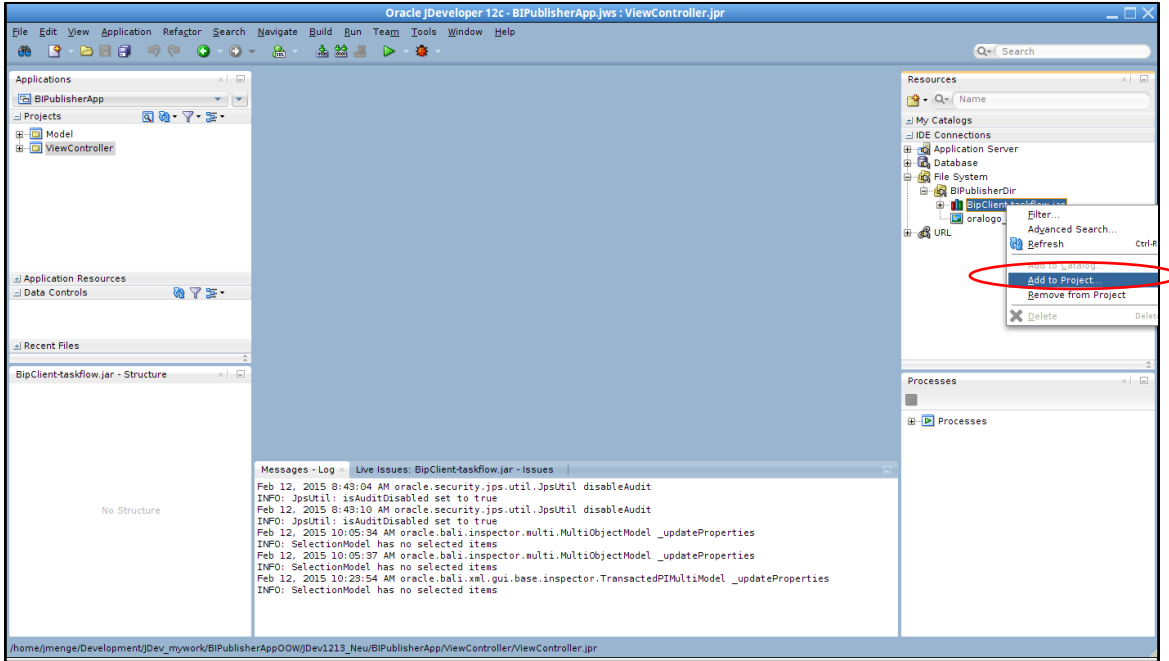
Figure 10 - Adding ADF library to the ViewController project

(1.4)   Save all by pressing the floppy disks symbol in the top menu of JDeveloper.

(2)　　　In this step we will integrate the ADF task flow from the library in the page. At runtime the task flow will be executed in a region as part of the page. Because the task flow is parameterized we have to provide values for these parameters.

(2.1)　　Follow these steps

Open the page *runBIP.jsf* in the workspace in Design View.

Select the window *Resources*.

Drag&drop the *bip-taskflow* into the facet *center* of the page.

Select the option *Region* to display the task flow as a static region inside the page.
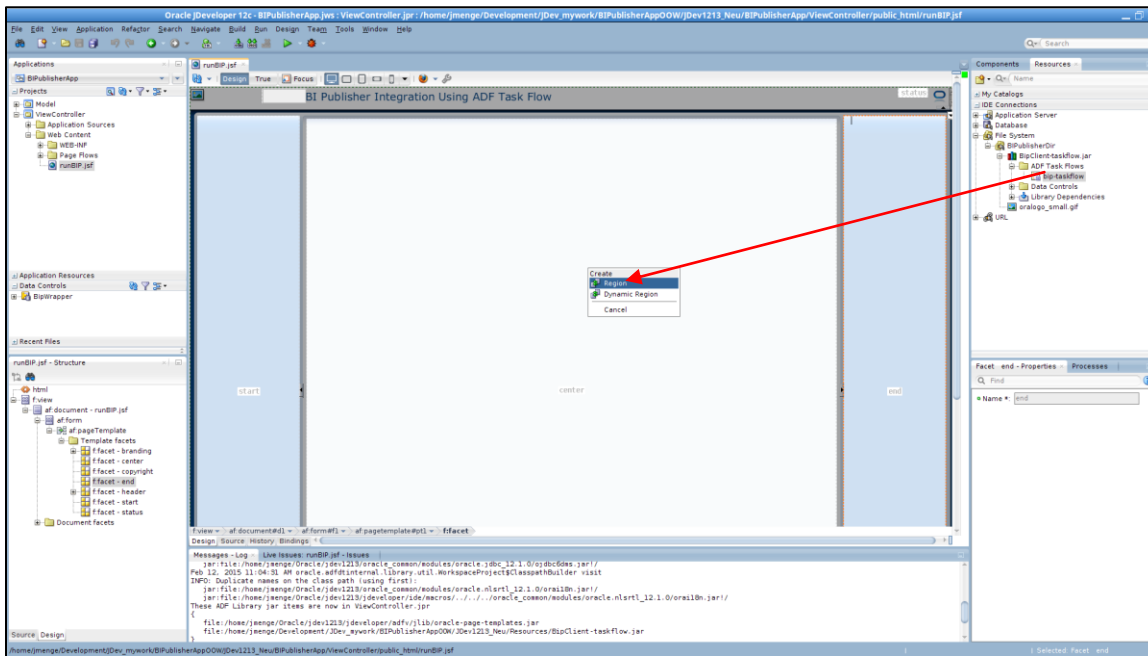
Figure 11 - Drag&Drop BIP task flow into the page

In the dialogue *Edit Task Flow Binding* we have to provide values for the mandatory parameters of the task flow.
Enter the following values:

Name	Value	Comment
BipAdminUserName	\<admin user name\>	BI Publisher administrator
BipAdminPassword	\<admin password\>	Password of administrator
BipUser	\<BIP user name\>	BI Publisher user. We use a static value here but it can be derived from the security context of the current user.
BipFolderPath	\<BIP folder\>	*BI Publisher folder*

The example actually utilizes BIP *impersonate* capabilities, as only the administrator credentials are authenticated, and once successful, admin impersonate the privileges of the user defined in parameter BipUser to fetch the reports he has access too.
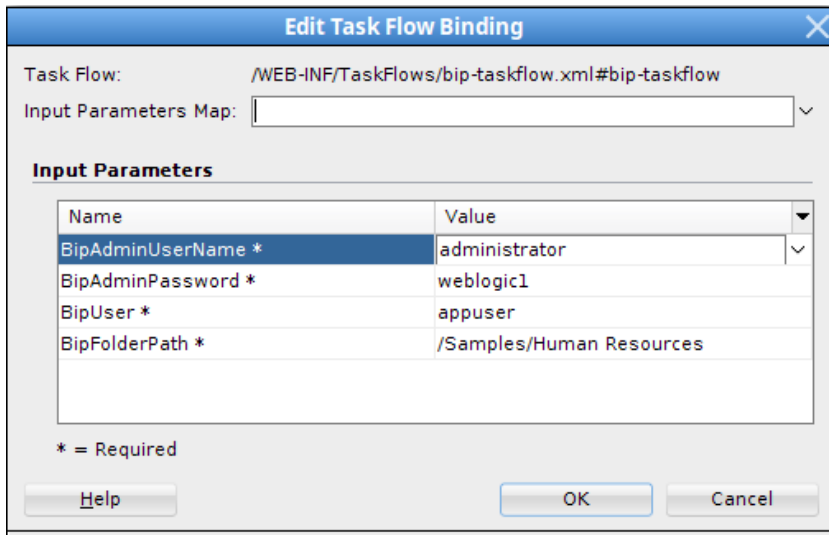
Figure 12 - Providing parameters for the BIP task flow

As a result a region is created inside the page but cannot be rendered at design time because the content is provided by the ADF Task Flow from the reusable library.
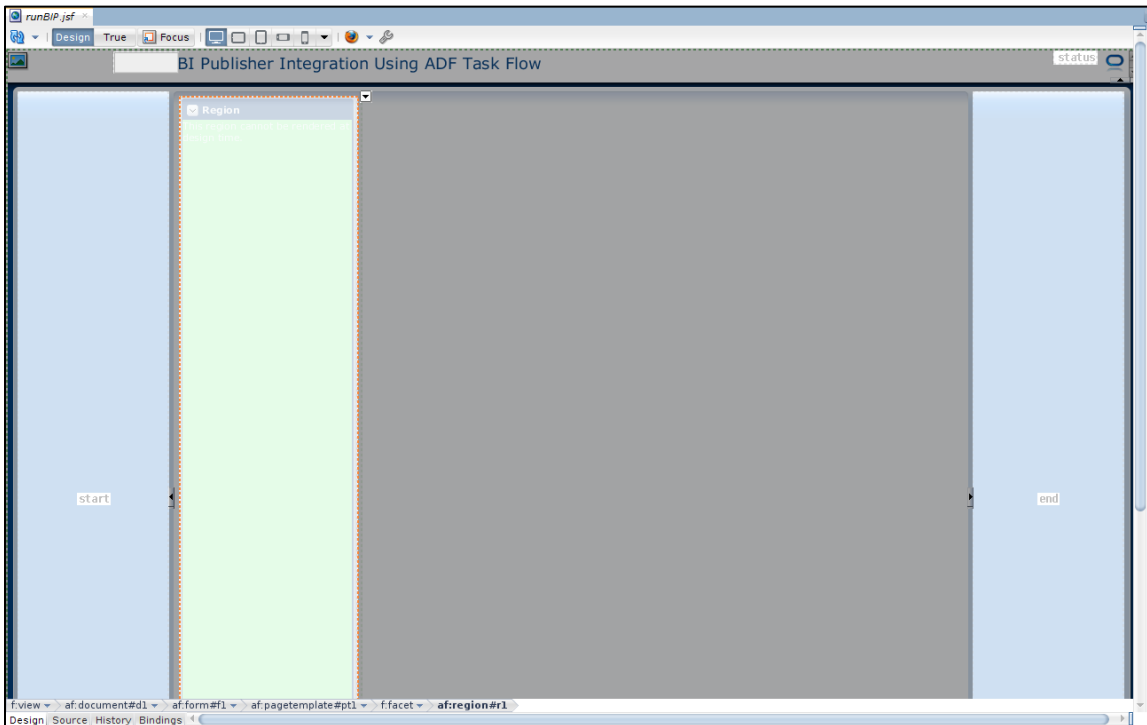
Figure 13 - Region inside the page

(2.2)   Save all by pressing the floppy disks symbol in the top menu of JDeveloper.

Run and test the ADF Application

(1)   In this section we will run and test the application using the integrated WebLogic Server in Oracle JDeveloper.

(1.1)   To see what happens open the window *Processes*. If not already visible open the window *Processes* by selecting *Window => Processes* from the top menu of JDeveloper.
Normally there should be no running processes at this time.
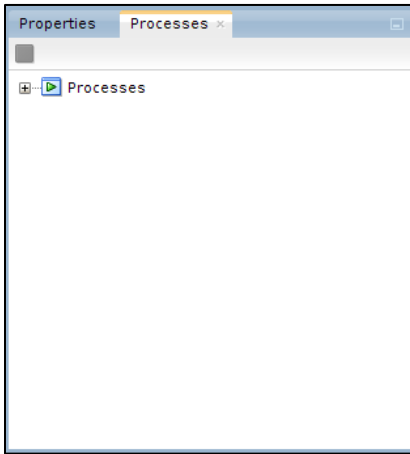
Figure 14 - Window Processes

(1.2)   Open the file *adfc-config.xml* in the *Diagram View*. Select the view *runBIP* and choose the option *Run* from the context menu.
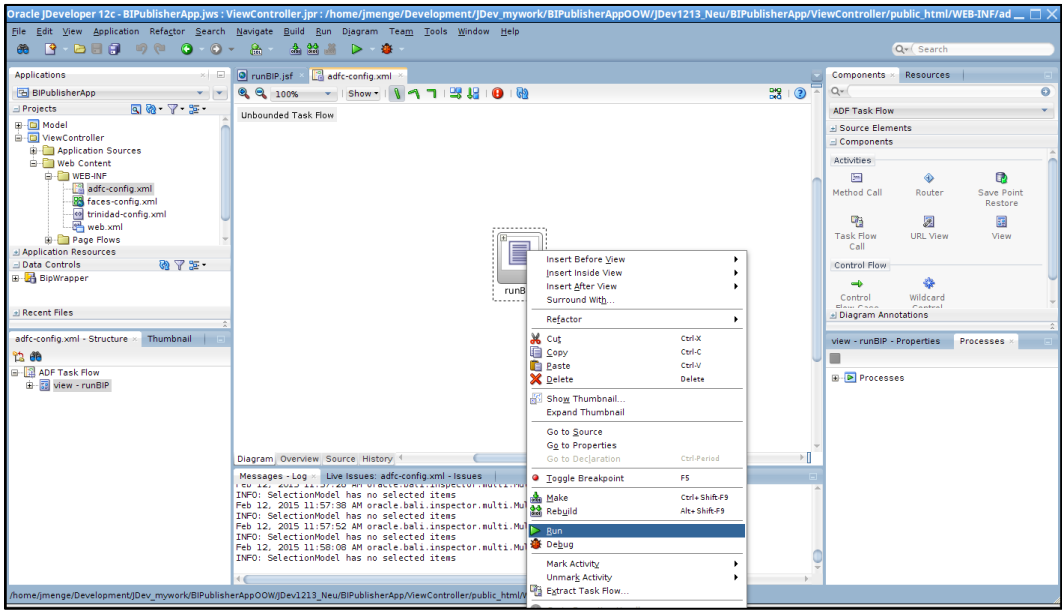
Figure 15 - Run page from adfc-config.xml

(1.3)   View in the *Processes* window.

First the integrated WebLogic Server will be started.

After WLS was succesful started our application will be deployed and started.
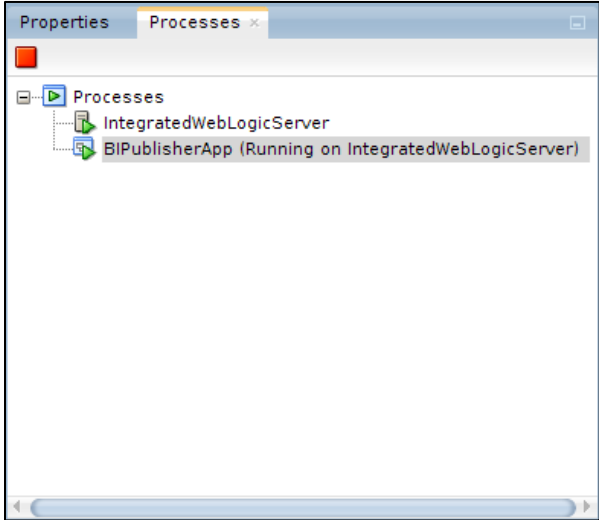
Figure 16 - Processes window after starting application

(1.4)   A browser window should be opened with our page. During the next steps we will stay on this page. Only the content of the embedded region is updated by a partial page refresh in the browser.

The first list-of-values for the *Report Name* displays the available reports from the specified folder.

When a report is selected the second list-of-values for the *Template* will be updated automatically to show the templates which are available for that report.

When a template is selected the third list-of-values for the *Format* will be refreshed automatically to show the formats which are allowed for that template.

If the selected report has associated parameters they will displayed too.

Every operation in the page triggers a web service call to the BI Publisher server, the delivered response will be displayed in the region.
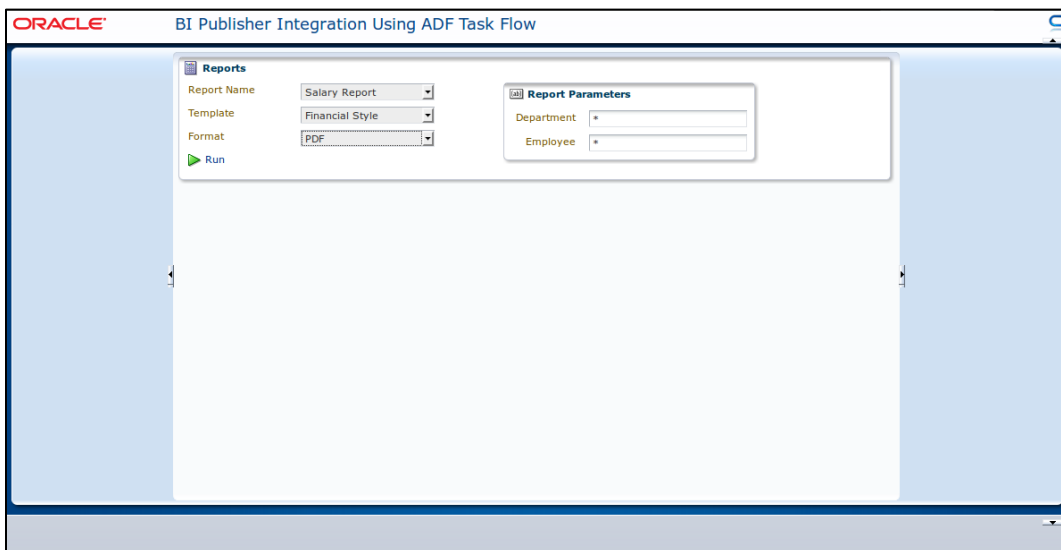
Figure 17 - Selection of report, template, format and report parameters

(1.5)   Now the report can be generated by pressing the green arrow button Run.

The report output will be produced on the server and downloaded to the client.

Depending from the selected format, the browser settings and your choice the result will be displayed in the browser (for example HTML), displayed using a helper application (for example PDF) or saved in the file system.

Figure 18 - Generated report (PDF)

Summary

This example shows how to integrate BI Publisher reporting capabilities in ADF applications. It especially shows the power of ADF task flows. The *bip-taskflow* completely encapsulates the required functionality to call BI Publisher web services. The advantages of this approach are obvious:

» The **developer for the model** (data control) provides an interface to BI Publisher server. He needs to know the web service API of BI Publisher and is responsible for the implementation.

» The **UI developer** can simply consume the task flow in his application and does not need to know the underlying web service implementation.

» The **task flow** can be used in all ADF applications which require this reporting functionality by simply adding the task flow as a library. Through parameterization the task flow can be reused in a very different context.

There are many ways this example can be extended:

» Specific **report parameters** like employee id or department number are implemented in the example as input text fields. It would be more comfortable to offer a list-of-values to the end user to select the values from. There are appropriate web service methods available to fetch this information from BI Publisher server

» There could be additional **scheduling parameters** to deliver a report immediately or at a certain time using mail or a printer.

» There are very specific **report formats** (PDF/Z and interactive) which are not supported in the example. It could also be useful to provide a **file extension** along with the file name when generating a report to allow saving the report as a file.

CONNECT WITH US

blogs.oracle.com/oracle

facebook.com/oracle

twitter.com/oracle

oracle.com

Oracle Corporation, World Headquarters

500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries

Phone: +1.650.506.7000
Fax: +1.650.506.7200

Hardware and Software, Engineered to Work Together

Oracle is committed to developing practices and products that help protect the environment