Oracle Forms & a Service Oriented Architecture (SOA).

A Whitepaper from Oracle Inc.
June 2007
# Table of Content

OVERVIEW...........................................................................................................................................5

HOW CAN FORMS BE PART OF A SERVICE ORIENTED ARCHITECTURE?...........................5

THE THREE AREAS..........................................................................................................................5

CALLING OUT FROM ORACLE FORMS............................................................................................6

USE OF EXTERNAL SERVICES.........................................................................................................6

CALLING JAVA.....................................................................................................................................6

Web Services.......................................................................................................................................11

Basic flow............................................................................................................................................11
Identifying the WSDL file..................................................................................................................11
Making the proxy...............................................................................................................................11
Package it all up................................................................................................................................12
Import into Forms.............................................................................................................................12
What if the Web service is secured with a password?......................................................................13

Business Process Execution Language (BPEL)..............................................................................14
An example..........................................................................................................................................14

EXPOSING FORMS BUSINESS LOGIC TO THE OUTSIDE WORLD........................................18

OVERVIEW..........................................................................................................................................18

A SIMPLE EXAMPLE........................................................................................................................18

USING THE APPLICATION SERVER'S INFRASTRUCTURE............................................................22

Enterprise Manager............................................................................................................................22
Single sign-on server...........................................................................................................................25
Enterprise User Security....................................................................................................................26
Switching it on....................................................................................................................................26
Defining users in OID..........................................................................................................................28

CONCLUSION.......................................................................................................................................31
TABLE OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIGURE 1</td>
<td>THE JAVA IMPORTER MENU ITEM</td>
<td>7</td>
</tr>
<tr>
<td>FIGURE 2</td>
<td>JAVA IMPORTER DIALOG</td>
<td>7</td>
</tr>
<tr>
<td>FIGURE 3</td>
<td>BASIC FLOW</td>
<td>11</td>
</tr>
<tr>
<td>FIGURE 4</td>
<td>WEB SERVICE PROXY WIZARD</td>
<td>12</td>
</tr>
<tr>
<td>FIGURE 5</td>
<td>BPEL PROCESS FLOW</td>
<td>15</td>
</tr>
<tr>
<td>FIGURE 6</td>
<td>APPLYING FOR A LOAN</td>
<td>16</td>
</tr>
<tr>
<td>FIGURE 7</td>
<td>APPROVING THE LOAN</td>
<td>16</td>
</tr>
<tr>
<td>FIGURE 8</td>
<td>CHOOSING THE BEST OFFER</td>
<td>17</td>
</tr>
<tr>
<td>FIGURE 9</td>
<td>EXAMPLE FORM</td>
<td>18</td>
</tr>
<tr>
<td>FIGURE 10</td>
<td>SCOTT IS LOGGED ON AND HE IS NOT THE PRESIDENT SO HE CANNOT SEE KING'S SALARY</td>
<td>20</td>
</tr>
<tr>
<td>FIGURE 11</td>
<td>COPYING A FUNCTION TO THE DATABASE</td>
<td>21</td>
</tr>
<tr>
<td>FIGURE 12</td>
<td>FORMS EM HOME PAGE</td>
<td>22</td>
</tr>
<tr>
<td>FIGURE 13</td>
<td>EM USER SESSIONS PAGE</td>
<td>23</td>
</tr>
<tr>
<td>FIGURE 14</td>
<td>EM CONFIGURATIONS PAGE</td>
<td>23</td>
</tr>
<tr>
<td>FIGURE 15</td>
<td>EM TRACE CONFIGURATIONS PAGE</td>
<td>24</td>
</tr>
<tr>
<td>FIGURE 16</td>
<td>EM ENVIRONMENT PAGE</td>
<td>24</td>
</tr>
<tr>
<td>FIGURE 17</td>
<td>EM JVM CONTROLLERS PAGE</td>
<td>25</td>
</tr>
<tr>
<td>FIGURE 18</td>
<td>EM UTILITIES PAGE</td>
<td>25</td>
</tr>
<tr>
<td>FIGURE 19</td>
<td>SSO FLOW</td>
<td>26</td>
</tr>
<tr>
<td>FIGURE 20</td>
<td>SWITCHING ON SSO</td>
<td>27</td>
</tr>
<tr>
<td>FIGURE 21</td>
<td>SINGLE SIGN-ON EXAMPLE</td>
<td>27</td>
</tr>
</tbody>
</table>
OVERVIEW

More and more businesses are looking to the principles of Service Oriented Architecture (SOA) to align their business and I.T. needs. The ability to build services modeled on business functions, reuse and orchestration of common, loosely coupled services and the agility associated with working with modular services, built on recognized standards, are attractive options.

Oracle Forms has been very successful in the market place but has traditionally been a monolithic tool. You either used Forms (and Reports and possibly Graphics) and only Forms or you didn't use it. In the new world of disparate and distributed services making up much of new development Forms has been changed from being monolithic to being part of a Service Oriented Architecture.

How can Forms be part of a Service Oriented Architecture?

Recent versions of Oracle Forms has gained functionality that makes it possible to integrate existing (or new) Forms applications with new or existing development utilizing the Service Oriented Architecture concepts.

With its support for Java and its integration into SOA, Forms provides an incremental approach for developers who need to extend their business platform to JEE. This allows Oracle Forms customers to retain their investment in Oracle Forms while leveraging the opportunities offered by complementing technologies.

But how do you actually integrate Forms with a Service Oriented Architecture. How can Forms be a part of SOA? This whitepaper is meant to shed some light on this topic.

THE THREE AREAS

There are three areas where Oracle Forms can be integrated with a Service Oriented Architecture:

- **Use of external service**
  
  With functionality recently added to Forms it is now possible to call from Forms to Java making it feasible to use Web services and BPEL processes.
• **Exposure of Oracle Forms business logic to the outside world**

In a world of distributed applications, Forms code might need to be moved out of Forms and into a place where it can be used by other applications. This section covers how to achieve that.

• **Using the Application Server's infrastructure.**

Oracle Forms coexists and integrates with Oracle's Applications Server's infrastructure functionality. Forms' integration with Oracle Single Sign-on and Enterprise Manager is covered in this section.

## CALLING OUT FROM ORACLE FORMS

### Use of external services

Oracle Forms use of external services hinges on recent functionality regarding Java integration. Oracle Forms can call out to Java on the file system. It can make use of Java beans and its native screen widgets can be customized with custom Java code (the Pluggable Java Component architecture). In this section we are going to touch on the functionality of calling out to Java code residing on the file system.

Once that functionality was put in place Oracle Forms was able to call all kinds of external services such as Web services and be part of a BPEL process flow. But let's start at the beginning.

### Calling Java

The functionality that makes it possible to call out to Java on the file system is called the Java Importer. It is incorporated into the Forms Builder and takes a class as an input and creates a PL/SQL package that acts as a wrapper around the Java class making it possible to call Java code from a PL/SQL trigger or function/procedure.
The Java Importer scans the local machine in the directories specified in the Registry variable FORMS_BUILDER_CLASSPATH and finds all Java class files. The Importer can scan JAR files only one level deep. It cannot find class files in JAR files that are in turn in a JAR file.
Using the new built-in packages `ora_java` and `jni`, the Importer will take a class definition like the following one and make a PL/SQL wrapper package.

```java
public class Dates {
    public static void main(String[] args) {
        System.out.println(theDate());
    }
    public static final String theDate() {
        return new Date().toString();
    }
}
```

The package will be created with the name of the class. Each public Java function will have a corresponding PL/SQL function and each Java function returning `void` will result in a corresponding PL/SQL procedure:

```sql
PACKAGE BODY dates IS
    args   JNI.ARGLIST;
    -- Constructor for signature ()V
    FUNCTION new RETURN ORA_JAVA.JOBJECT IS
        BEGIN
            args := NULL;
            RETURN (JNI.NEW_OBJECT('formstest/dates', '()V', args));
        END;
    -- Method: main ([Ljava/lang/String;)V
    PROCEDURE main( a0    ORA_JAVA.JARRAY) IS
        BEGIN
            args := JNI.CREATE_ARG_LIST(1);
            JNI.ADD_OBJECT_ARG(args, a0, '[Ljava/lang/String;');
            JNI.CALL_VOID_METHOD(TRUE, NULL, 'formstest/dates', 'main', '([Ljava/lang/String;)V', args);
        END;
    -- Method: theDate ()Ljava/lang/String;
    FUNCTION theDate RETURN VARCHAR2 IS
        BEGIN
            args := NULL;
            RETURN JNI.CALL_STRING_METHOD(TRUE, NULL, 'formstest/dates', 'theDate', '()Ljava/lang/String;', args);
        END;
BEGIN
    NULL;
END;
```
Java classes declared static, classes that are always instantiated with one and only instance in a Java application, will have no new functions generated.

The package code can subsequently be used to, at runtime, call out to the Java code and return not just scalar values as in this example but complex data structures like arrays and Java objects. There are functions in the ora_java built-in PL/SQL package to manipulate arrays and complex data types.

At runtime the class files needs to be accessible to the Forms runtime. To do that you add either the full path to the directory the class file is in or add the full path to the JAR file to the CLASSPATH environment variable in the default.env file (see the EM section below for more information).

It is also possible to import system classes and thus manipulate Java data types and extract data that can be used in PL/SQL.

One such example is the java.lang.Exception class. The following is a partial representation of the PL/SQL package code that is the result of importing the Exception class.

```plsql
PACKAGE BODY Exception_ IS
  args   JNI.ARGLIST;
  -- Constructor for signature
  -- (Ljava/lang/Throwable;)V
  FUNCTION new(
    a0  ORA_JAVA.JOBJECT) RETURN ORA_JAVA.JOBJECT IS
    BEGIN
      args := JNI.CREATE_ARG_LIST(1);
      JNI.ADD_OBJECT_ARG(args, a0,
        'java/lang/Throwable');
      RETURN (JNI.NEW_OBJECT('java/lang/Exception',
        '(Ljava/lang/Throwable;)V', args));
    END;
    -- Method: toString ()Ljava/lang/String;
    FUNCTION toString(
      obj   ORA_JAVA.JOBJECT) RETURN VARCHAR2 IS
    BEGIN
      args := NULL;
      RETURN JNI.CALL_STRING_METHOD(FALSE, obj,
        'java/lang/Exception', 'toString',
        '()Ljava/lang/String;', args);
    END;
  BEGIN
    NULL;
  END;
END;
```

‘Exception’ is a reserved word in PLSQL so the importer resolves a potential naming conflict by adding an underscore.

The resulting Exception_ class is used in this PL/SQL function shown in more depth later in this white paper when obtaining a currency conversion rate from a Web service.
The Exception_.toString call is used to get a character representation of an exception obtained from Java that is of a type PL/SQL cannot understand but Java can.

```java
function get_conversion_rate(currency_code varchar2) return number IS
    conv ora_java.jobject;
    rate ora_java.jobject;
    excep ora_java.jobject;
begin
    conv:=currencyConverter.new;
    rate:=currencyConverter.getrate(conv, 'USD', currency_code);
    return float_.floatvalue(rate);
exception
    when ora_java.java_error then
        message('Error: ' || ora_java.last_error);
        return 0;
    when ora_java.exception_thrown then
        excep:=ora_java.last_exception;
        message('Exception: ' || Exception_.toString(excep));
        return 0;
end;
```
Web Services

A Web service is a piece of code that can accept remote procedure calls using the HTTP protocol and an XML based data format called SOAP and return data in the form of XML to the originator. Web services are a major part of any Service Oriented Architecture and makes it possible to distribute business logic exposed as a service onto servers on the local network or even the Internet.

Forms cannot make use of a Web service directly but now that we know how to call out to Java we can make it happen. With a proxy made in Oracle's JDeveloper we can call any Web Service from Forms.

Basic flow

See this link to learn more about the WSDL file standard.

Identifying the WSDL file

A Web service is defined by a WSDL (Web Service Description Language) file. It has definitions on how to call the service and what you can expect as return data. Since a Web service is a network service, its definition is also a network resource. The administrator of the Web service you are interested in, should be able to tell you how to get the WSDL file URL. Web services may be published in Universal Description Discovery Integration (UDDI) registry which acts like a telephone directory of Web services.

Making the proxy

A proxy, in this context means a Java class that has been instrumented to know how to call a specific Web service. Oracle's JDeveloper can help us to make the proxy.

JDeveloper has a wizard that takes a WSDL file as input and creates a Java package that we can subsequently import into Forms, making it possible for our Forms application to call the Web service.
Once you have the WSDL file URL you can plug it into the JDeveloper Web Service Proxy Wizard. The screenshot below has a WSDL file already plugged in.

![Create Web Service Proxy](image)

**Figure 4: Web service Proxy Wizard**

The name of the service defined in this WSDL file is CurrConv and the wizard has picked that up. See this link for exact instructions on how to make a Web service proxy.

**Package it all up**

See this link to learn how to package the Web service proxy that you have created up so that you can import it into Forms.

**Import into Forms**

Once you have the proxy class you can import it into Forms and call its functions from PL/SQL code. The operative functions imported from the proxy are listed below:

```sql
package CurrencyConverter
    /* currconv3.mypackage.CurrConv3Stub */ IS
function new return ora_java.object;
function getRate(
    obj   ora_java.object,
    a0    varchar2,
    a1    varchar2) return ora_java.object;
end;
```
The `getRate` function is the function that we are after. It takes two currency symbols and returns a Java object that we know is a number of type float (the first argument is the class instance and will be described later).

The WHEN-BUTTON-PRESSED trigger code is listed below:

```sql
function get_conversion_rate(currency_code varchar2) return number IS
  conv ora_java.jobject;
  rate ora_java.jobject;
  excep ora_java.jobject;
begin
  conv:=currencyConverter.new;
  rate:=currencyConverter.getrate(conv, 'USD', currency_code);
  return float_.floatvalue(rate);
exception
  when ora_java.java_error then
    message('Error: ||ora_java.last_error);
    return 0;
  when ora_java.exception_thrown then
    excep:=ora_java.last_exception;
    message('Exception: ||Exception_.toString(excep));
    return 0;
end;
```

This code delivers the rate used in the application. It takes a currency symbol (in this example we only convert from US dollars) as an argument and returns a PL/SQL number which holds the rate.

We first declare three local variables of type JOBJECT (JOBJECT is a type defined in the ora_java package representing a Java object of any kind). The first line after `begin` fetches a reference to an instance of the proxy and stores it in the variable `conv`. The second line gets the rate from that instance. The return clause makes a number from the resulting rate Java object with the help of an imported Java system class, `java.lang.Float`. It has a call, `floatValue`, which converts a float to a number, which is what we need here (Forms cannot handle a Java float directly but it can handle a number).

The exception handler is reached if any of the calls out to Java causes (throws is the term used in Java) an exception or an error. In case of an exception we get the exception with the help of the `ora_java` package call `last_exception`. To actually see it, we call an imported `Exception_` package routine called `toString`.

**What if the Web service is secured with a password?**

This page has a good tutorial on how to change a Web service proxy to use authentication.
Business Process Execution Language (BPEL)

Now that we have established that Forms can call out to Java and use Web services, we can start looking at BPEL or Business Process Execution Language. BPEL is an emerging standard for orchestrating disparate and heterogenous business services into a process flow. Oracle offers a comprehensive BPEL solution and Forms can take part in a BPEL process flow as a manual process step or as an initiating step or both.

An example

In the following example Forms is going to act both as an initiator and as a manual process step. The process is that of a consumer loan. The business flow diagram below outlines what the process looks like.
The process first receives an application for a consumer loan. This step is performed in a Forms application that communicates with the BPEL server through either a Web service exposed by the BPEL server or by way of a Java interface supplied by Oracle as a part of the BPEL server.

Figure 5: BPEL Process Flow
The screenshot above shows the application that the fictitious user Dave uses to send in his application for a car loan. When the user clicks the Submit button the Forms application sets in motion the BPEL process flow described earlier. The process fetches Dave's Social Security Number and then gets his credit rating. As the developer of the application that Dave uses we don't have to know how this is achieved. That is up to the developer of the BPEL process. We just need to know how to kick off the process.

After having fetched Dave's credit rating the BPEL process collects two offers from two different loan vendors, Star Loans and United Loans. United Loans uses an automated process that we, in this example, are not interested in. Star Loan on the other hand uses a manual process that is implemented in a Forms application.

The loan officer at Star Loan queries the BPEL process and sees that an application has arrived from Dave. He/she determines the appropriate interest rate and clicks the Approve button. This will kick off the next step of the process where Dave has the opportunity to select the best offer. He does that in the same Forms application he used earlier:
Dave sees the two loan offers in his application and can select the best offer by clicking the Accept button. This will again cause Forms to communicate with the BPEL server and causes the process to conclude.

Note the Refresh button. Forms could potentially poll the BPEL server with the help of a Forms timer. This application does not do that. Instead Dave has to manually query the BPEL process. Forms cannot yet (as of version 10.1.2) easily register an interest in a BPEL event and automatically be notified if input is needed from it. In version 11 of Forms we intend to have functionality in place that will make this much easier.

For more information on how to achieve BPEL integration and a working example, see this link.
EXPOSING FORMS BUSINESS LOGIC TO THE OUTSIDE WORLD

Overview

So far we have covered how Forms can call out to the outside world. What about the opposite? Can the outside world use existing Forms business logic? Perhaps as exposed as a Web service? Or could Forms business logic (which is written in PL/SQL) be called directly?

The answer is a qualified Yes. As a development tool, the fact that the user interface and the business logic are so closely integrated, makes the development of Forms very simple and intuitive. However, this tight integration of UI code and business logic makes the exposure of the pure business services to outside consumers, much more challenging.

It is however possible to move Forms PL/SQL code from Forms to the database and from there expose it to the outside world, either as a database procedure/function or as a PL/SQL Web service.

A simple example

In this simple example we have a block in Forms with a column labeled Salary that shows the total salary for the employees in the Emp table. The sum is calculated with a POST-QUERY trigger.

![Example form](image)

The trigger calls a local (it is executed by Forms rather than the database) PL/SQL function thus:
function calc_total_sal return number is
  total number;
  l_mgr number;
  l_mgr_name varchar2(30);
  usr varchar2(200):=
    get_application_property(username);
begin
  select mgr into l_mgr
    from emp
    where empno=:emp.empno;
  select ename into l_mgr_name
    from emp
    where mgr is null;
  if (l_mgr is not null) then
    select sal+nvl(comm,0) into total
    from emp
    where empno=:emp.empno;
  elsif (usr<>l_mgr_name) then
    total:=-1;
  elsif (usr=l_mgr_name) then
    select sal+nvl(comm,0) into total
    from emp
    where empno=:emp.empno;
  end if;
  return total;
end;

The business logic implemented in the function is this: If the employee whose salary is being calculated is not the President, calculate his total salary by adding columns sal and comm together, taking into account that the comm column can potentially be null. If the employee row is the President's and the current user is not the President return -1 otherwise return the total salary for the President (nobody but the President can see the salary of the President of the company).

To use this code from the database we need to refactor it. The database does not understand references to any Forms objects nor does it understand any of the Forms specific PL/SQL built-ins. In this case we have a call to a Forms built-in, namely get_application_property. We also have a reference to the empno field. If we take them out and pass them in as parameters the function now looks like this:

function calc_total_sal(l_empno number, usr varchar2) return number is
  total number;
  l_mgr number;
  l_mgr_name varchar2(30);
begin
  select mgr into l_mgr
    from emp
    where empno=l_empno;
  select ename into l_mgr_name
    from emp
    where mgr is null;
if (l_mgr is not null) then
    select sal+nvl(comm,0) into total
    from emp
    where empno=l_empno;
elsif (usr<>l_mgr_name) then
    total:=-1;
elsif (usr=l_mgr_name) then
    select sal+nvl(comm,0) into total
    from emp
    where empno=l_empno;
end if;
return total;
end;

Note that the variables usr and l_empno are now external and have to be passed in. The POST-QUERY trigger has to change accordingly, to say;

:emp.total_sal:=calc_total_sal(:emp.empno,
    get_application_property(username);
but after that it will continue to function as before.

Figure 10: Scott is logged on and he is not the President so he cannot see KING's salary

Moving a PL/SQL procedure from a form definition file to the database can be achieved in the Forms Builder by dragging and dropping the PL/SQL unit between a form module and a database node in the Forms navigator.
Now that we have the Forms PL/SQL code in the database, we can leverage its business logic in any application that can call a database function.

With the help of JDeveloper we can now also make a Webservice out of the code, making it possible for environments that cannot directly call into the database but which can call a Web service, to leverage legacy Forms code.

See this web page for more information on how to use JDeveloper to make a Web service from database PL/SQL code.
**USING THE APPLICATION SERVER'S INFRASTRUCTURE**

When Forms becomes a part of a larger setting it needs to be able to participate in application server wide functions such as maintenance and management and user authentication. It doesn't make much sense to have one application use its own authentication scheme and all the others use another scheme. In versions 9 and 10 Forms is a full member of the Application Server infrastructure and is automatically configured to be able to use both the Single Sign-on server (Oracle OID and SSO) and to be managed thru Oracle Enterprise Manager.

**Enterprise Manager**

As part of the Oracle Application Server platform, Oracle Forms applications can now be managed remotely through Oracle Enterprise Manager's Application Server Control running in a browser.

This is the screen that meets the administrator when he logs onto the Application Server Control console:

![Forms EM Home page](image)

In this overview screen the administrator can monitor the over-all status of the system and critical metrics such as CPU and memory usage for Forms and its main components.

---

Oracle Forms and SOA. Page 22
In this next screen, called User Sessions, each user's session is listed with crucial metrics such as its CPU and memory usage, IP address and username and trace settings. The configuration section used when starting the application is also shown. If tracing is turned on from here the trace log can be viewed from this console. Individual user sessions can be terminated from here.

Figure 13: EM User Sessions page

Figure 14: EM Configurations page
In this section you can create, edit and delete configuration settings in three different configuration files: the formsweb.cfg which is the main configuration file for a Forms installation, the ftrace.cfg which is the trace facility configuration file and the Registry.dat file which governs the font mapping. Here is a screenshot of the ftrace section:

![Figure 15: EM Trace Configurations page](image1)

![Figure 16: EM Environment page](image2)

In this Environment screen the administrator can manipulate environment settings stored in the default.env file that pertain to Forms:
JVM Controllers are used to reduce memory requirements when calling Java from the file system and they are controlled from this screen.

The last screen is used for additional functionality not yet accessible from other screens.

**Single sign-on server**

Changing Forms to use the Single Sign-on server rather than its own database based user authentication method is a matter of setting a flag in a configuration file.
Once Forms is set up to use SSO, Forms users who are logged on because they have done so in their Forms application need not reauthenticate themselves when they log in to a Portal or a Java application.

SSO-based authentication happens in 5 steps. In step 1 the user supplies his credentials (username and password). Instead of calling the database to authenticate the user, Forms calls the SSO server who in turn queries the OID/LDAP server to see if the user has enough privileges. If so, it returns the actual database username and password that will be used by Forms to log on. During the session Forms can access data about the logged in user from the OID repository thru a PL/SQL API.

The same single sign-on instance can of course be used by other applications served from the same application server so users need not be stored in multiple places and can be authenticated only once, namely the first time they log in to any of the applications making use of the same SSO Server.

**SSO & OID**

**Figure 19: SSO flow**

**Enterprise User Security**

The database can also use the Single Sign-on server as its authentication scheme with functionality called Enterprise User Security. This makes it possible to store authentication information in a single place, both for access to the database thru an application and thru direct means, such as SQL*Plus or the new SQL Developer.

**Switching it on**

Making Forms use the Single Sign-on server is as easy as switching a switch in the formsweb.cfg file.
Figure 20: Switching on SSO

Here we are setting the ssoMode switch in Enterprise Manager's Application Server Configuration section to true. This is all that is needed to enable SSO in a default Forms installation. When a user starts a Forms application he/she will be met by this SSO login screen instead of by the normal Forms login window:

![Sign In Form]

Figure 21: Single Sign-on example
Defining users in OID

SSO users must be defined in OID. That can be done in Portal. Log in as orcladmin after clicking on the link in the Application Server welcome screen that is highlighted in the following screenshot:

![Figure 22: Loggin on to Portal](image)

Click the Administer tab when you arrive at this screen:

![Figure 23: Administer page](image)

This will take you to a part of the Portal builder that has a user interface for OID management. To create a new user, click the link in the User portal in the top right hand corner that is highlighted in the following screenshot:
Doing that will open up this screen where you can specify all the data you will need about the user, such as First and Last name, username, e-mail, permissions and roles.

![Create New Users](image)

**Figure 24: Create new user**

A mandatory part (for Forms users) that is not entirely obvious is the Resource Access Information part:

![New User page](image)

**Figure 25: New User page**

A mandatory part (for Forms users) that is not entirely obvious is the Resource Access Information part:
This information will be used by Forms (and Reports incidentally) when logging on to the database. The user will be authenticated by the information specified in this big screen but the actual log-in will be done with the account specified in this section.

To create a new resource click the Create Resource button and fill in the name of the resource in this screen:

The resource type should be OracleDB. Click the Next button to get to this screen:
CONCLUSION

This whitepaper has covered the topic of integrating Oracle Forms with a Service Oriented Architecture. We had a look at how Forms can call out to Java, thus making it possible to make use of Web services and process flows orchestrated by BPEL. We touched on the process of exposing existing Forms code to SOA processes by refactoring the code and moving it to the Oracle Database. Finally we explored how Forms can take advantage of the remote management capabilities or Oracle Enterprise Management (EM) and of centralized user authentication by way of Oracle Identity Management (SSO and OID).