professional
it software &
services

Implementing Web Services in Oracle Database Applications

How to integrate legacy Oracle Forms applications in the modern Web development world

Technical Article, February 2011

1. Introduction

Data managing applications exist since the beginning of the IT industry, supporting gradually most of our business processes and constantly evolving to adapt to the ever changing business needs. In all this quest of quickly developing, adapting, fixing and patching, the typical Oracle database application has grown bigger and bigger in time, consisting now of a mix of new and old components, written in various programming languages and having different architectures. Our most challenging task is making the most of all these components by helping them inter-operate and function as a whole modern application that fully supports the business requirements - not only today but also in 10 years time or even more.

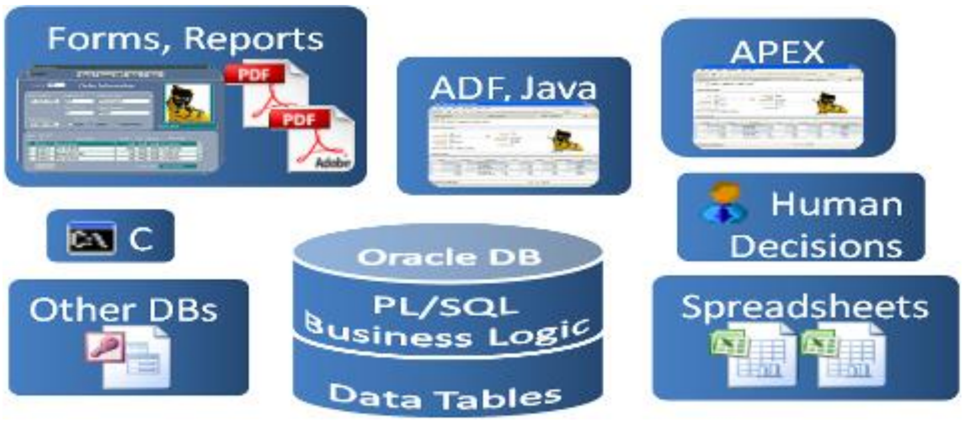


Figure 1: Oracle Database application

The current paper provides an analysis of the possibilities we have to make legacy Oracle Forms applications become a part of the modern Web development world. As the service-oriented architectures are gaining increased significance, we will assess the means we have to communicate with external applications by implementing Web services.

2. A few words about Oracle Forms applications

Oracle Forms is a solid and mature framework, the mostly used environment for developing Oracle database applications for the last decades. Its architecture has been designed specifically for rapid and easy development of complex database applications. However, 25 years ago, when the Forms architecture has been designed, the need of interoperability and communicating via Web services was not as critical as nowadays. This is why modernizing now Oracle Forms applications to be able to inter-operate with external applications is not a straightforward task:

- As **Web services consumer**, unlike modern applications developed with Oracle ADF or APEX, Forms applications do not have own mechanisms to establish a direct connection to Web services. However, they can indirectly access Web services by using a proxy or through the database Web services utilities. We will analyze both solutions in detail and see which one suits better to our requirements.

- As **Web services provider:** If we want, however, to publish Forms functionalities (= business logic) as Web services, we will need to transfer the corresponding code to the database and use the Oracle database capability to act as a Web service provider. Forms alone cannot have this role, because of its compact architecture. We will see how to perform this and why moving the business logic to the database is increasing the degree of code reuse and easing the future migrations of our Forms applications to modern frameworks.

3. Consuming Web services

As discussed before, we have two possibilities for accessing external Web services from Oracle Forms applications: directly from Forms by using a Web services proxy, or from the Oracle database using its Web service utilities.

3.1. Calling Web services from Oracle Forms

How can Oracle Forms access external Web Services? Well, Forms is not able to communicate directly with Web services but can communicate with Java classes inside a Web services Proxy. The Proxy will act as a mediator, being able to establish a connection to the Web service and communicate to its functionality, as depicted in Figure 2.

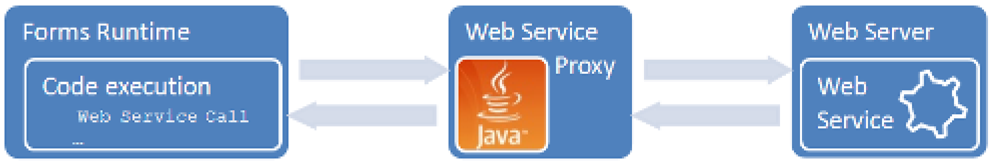


Figure 2: The communication between Forms and Web service via Proxy

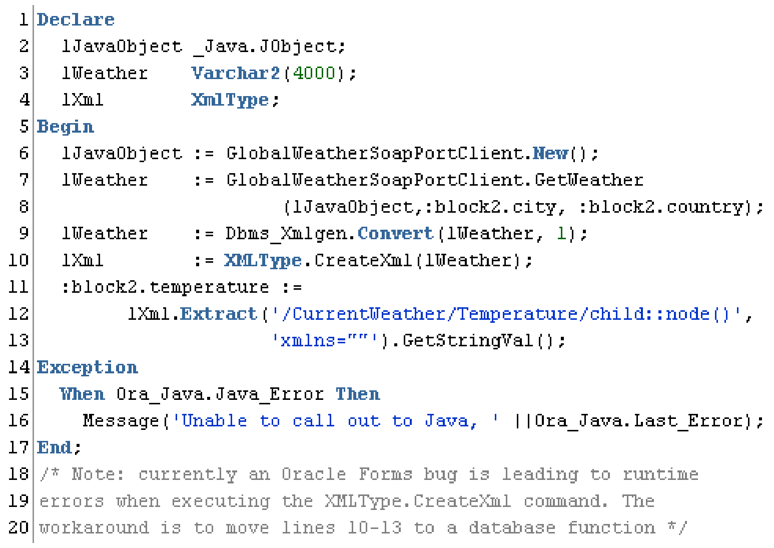
To implement such a Web service calling mechanism in a Forms application we need to perform a series of steps:

- **Generating the Web Service Proxy component**
  This can be easily made using the Oracle JDeveloper wizard. We need to specify there the URL where the Web service WSDL resides. This is a XML-document containing information about Web service operations, addresses where these operations can be called, communication messages and others. Important here is using the correct Java version: when generating and compiling the Java files we should use the Java version available in Forms JRE.
- **Packing the Web Service Proxy to a Jar-file**
  This can also be done with Oracle JDeveloper by deploying the project containing the generated Web service Proxy as a Jar file.
- **Setting the FORMS_BUILDER_CLASSPATH environment variable**
  In the Registry Editor, we need to set the FORMS_BUILDER_CLASSPATH variable in the Oracle Home corresponding to the current Forms version. Its value data should be set to a location that

contains the generated Proxy JAR file. This way the Web service Proxy Java classes will be visible to the Forms Java Importer.

- **Importing the Java classes in the Forms application**
  Here we will use the Forms Java Importer to import to our Forms application the Proxy Java class that contains the methods communicating to the Web service. These classes are usually named like *WebServiceName*PortClient. The Java Importer will create in the Forms application a package specification and body that represent the PL/SQL interface for the imported Java class.
- **Setting the CLASSPATH environment variable for runtime**
  In the ENV file used by Forms application at runtime we have to add the Proxy JAR file location to the CLASSPATH environment variable. This setting is necessary to make the Proxy Java classes available to the Forms runtime.

Once we have implemented this mechanism in our Forms application, we are able to use the functionalities defined in the newly generated package and call the Web service.



```
 1 Declare
 2   lJavaObject _Java.JObject;
 3   lWeather    Varchar2(4000);
 4   lXml        XmlType;
 5 Begin
 6   lJavaObject := GlobalWeatherSoapPortClient.New();
 7   lWeather    := GlobalWeatherSoapPortClient.GetWeather
 8                    (lJavaObject,:block2.city, :block2.country);
 9   lWeather    := Dbms_Xmlgen.Convert(lWeather, 1);
10   lXml        := XMLType.CreateXml(lWeather);
11   :block2.temperature :=
12         lXml.Extract('/CurrentWeather/Temperature/child::node()',
13                      'xmlns=""').GetStringVal();
14 Exception
15   When Ora_Java.Java_Error Then
16     Message('Unable to call out to Java, ' ||Ora_Java.Last_Error);
17 End;
18 /* Note: currently an Oracle Forms bug is leading to runtime
19 errors when executing the XMLType.CreateXml command. The
20 workaround is to move lines 10-13 to a database function */
```

Figure 3: Forms code example for getting the temperature for a given location by calling the http://www.Webservicex.net/globalweather.asmx?wsdl Web service

3.1.1. The Synchronous vs. Asynchronous dilemma

The code in Figure 3 is an example of synchronous calling of a Web service. The Proxy is receiving the call from the Forms application and sending it further to the Web service. It afterwards receives the answer from the Web service and forwards it back to the Forms application. All this time the Forms resources are blocked (locked). The code execution is stopped until the Forms Runtime receives an answer from the Web service. This is why, when calling a synchronous Web service, we need to take the following into consideration:

- Do we need the Web service answer for continuing the code execution? Because if not, we may be able to us an asynchronous Web service.
- How much time do we estimate the Web service execution will take? Can we afford to block the application resources for this period?

- Considering the resources that are blocked during Web service execution: how often do we need them for other processes in our application?

The last two questions are really important: we need to avoid a cascade locking of all the database sessions trying to access the resources blocked during Web service execution. The solution to this: Asynchronous Web services.

A possible architecture for asynchronous calls would additionally include a one-way BPEL process, the Advanced Queuing (AQ) database functionality and the new Event object we have in Forms 11g.

3.1.2. Using Forms 11g Event object

In the new architecture the Proxy is not mediating the communication between Forms and Web service, but between Forms and the new BPEL process. And because this BPEL process is one-way (shut and forget), the Web service Proxy will not wait for the BPEL process execution. Upon receiving the request confirmation from the BPEL process, the Proxy will allow Forms to execute the remaining code. The BPEL process will forward the call to the Web service and wait for response. After receiving the response, the BPEL will return it as a message to the queue referred by an Event object in the Forms application. We have here more possibilities for the BPEL process to send the message:

- Direct response transmission using an AQ adapter
- Indirectly through a DB adapter – depending on the way the DB adapter has been defined we have the possibility to execute additional actions in the database before sending the message further to the queue, using a stored trigger, function or procedure.

When the message is placed in the queue, AQ notifies the Forms Runtime. The When-Event-Raised trigger will be executed at the next ping received from the applet, processing the received message.[1]

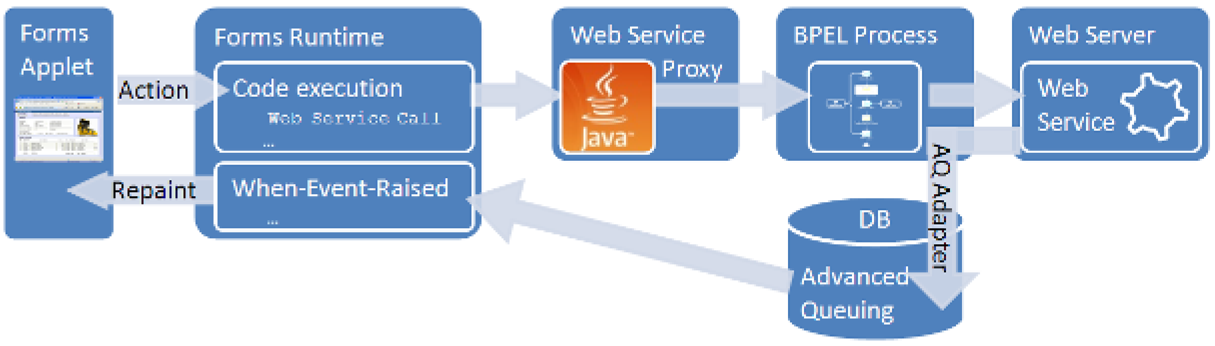


Figure 4: The communication between Forms and Web service via Forms11g Event

A few words on the programming style: because the Web service execution may take time and the Forms application may loose control over the started processes, it is recommended that the BPEL process to regularly send notifications on the execution status to the Forms application. More, if the

[1] More details on the Event object at: http://www.oracle.com/technology/sample_code/products/forms/11g/aqinteg/lab1/index.htm#o

Web service could notify from time to time the BPEL process about the process stage, the BPEL process would take these messages and forward them to the Forms application as well.

3.2. Calling Web services from the Oracle database

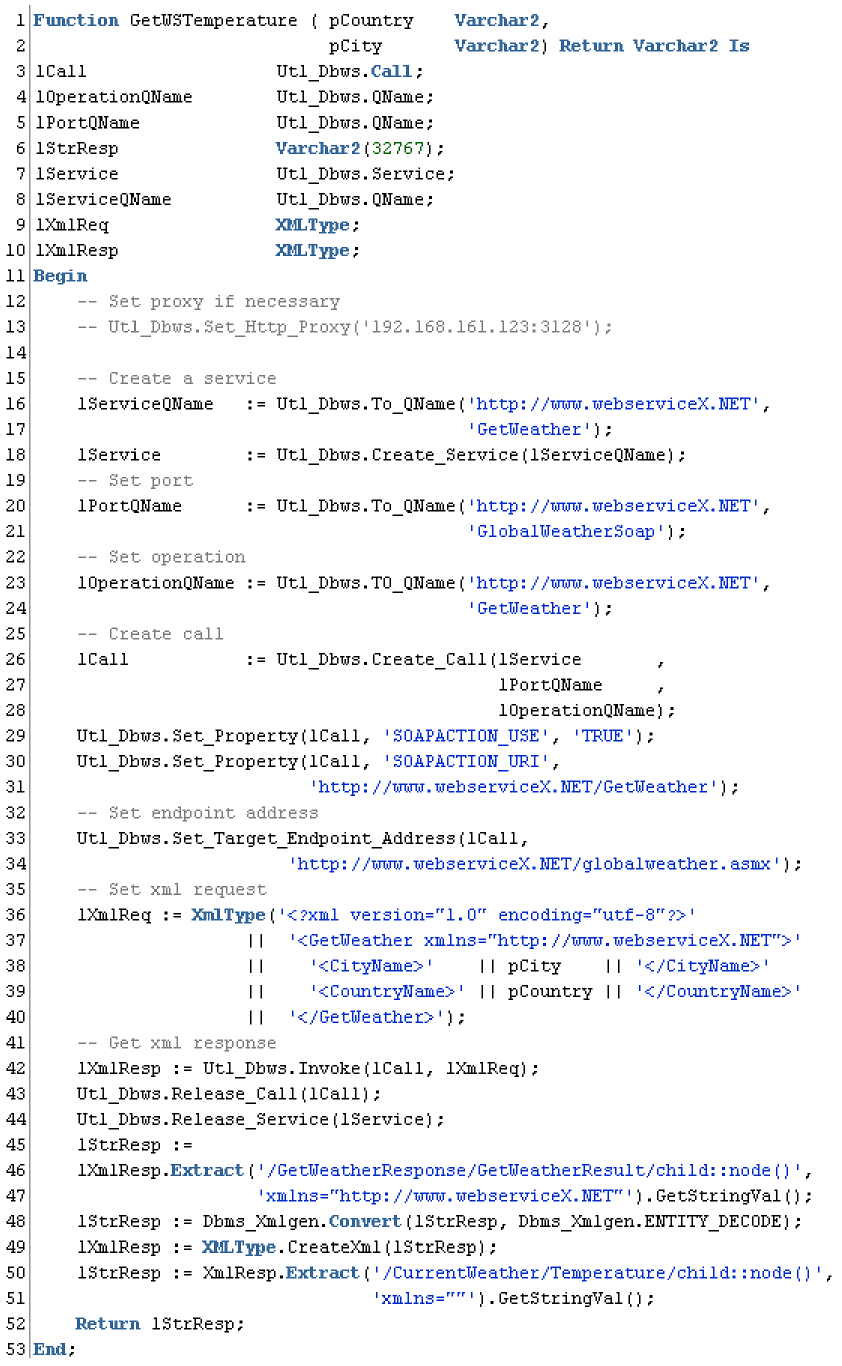
We have seen how we can call Web services from Forms. But what if functionality stored in the database like a piece of business logic migrated from the Forms application needs to call itself a Web service? How can we replace the previously discussed proxy Web service call? Oracle offers several possibilities to perform this:

- **Java solution**: One solution is deploying a Java Web service client to the database. However, deploying the Java Web service dependent classes to the database may lead to incompatibilities with some Java classes used by Oracle DB and could affect the database stability[2].
- **`UTL_HTTP`** – available starting with Oracle 7.3.4, is offering the possibility to execute HTTP callouts from the database. It can also be used for communicating to a Web service by sending a HTTP request that contains a SOAP request message for Web service and receiving a HTTP response with the SOAP response message.
- **`UTL_DBWS`** – available starting with Oracle 10g. Being focused on making SOAP calls, it is better equiped for Web services compared to UTL_HTTP. The functions and procedures defined in this package represent wrappers for the JAX-RPC dynamic Invocation Interface. A stub is dynamically generated each time we access a Web service via UTL_DBWS. Note: Although UTL_DBWS comes pre-installed with Oracle 10g, its Callout Utility is not immediately functional and needs to be installed afterwards[3]. In Figure 5 we can see an example of using the UTL_DBWS package to call an external Web service for getting the temperature for a specific location:

The last two PL/SQL solutions have the advantage to integrate perfectly in an Oracle database application. UTL_DBWS is newer and recommended by Oracle, as new features, bug fixes and patches are available only for the DBWS Callout Utility as far as SOAP calls are concerned.

[2] Database Web Service Callout Utility 10.1.3.:
http://www.oracle.com/technology/sample_code/tech/java/jsp/callout_users_guide.htm

[3] Details on installing Oracle Callout Utility: http://www.oracle-base.com/articles/10g/utl_dbws10g.php



```
Function GetWSTemperature ( pCountry    Varchar2,
                            pCity       Varchar2) Return Varchar2 Is
lCall                   Utl_Dbws.Call;
lOperationQName         Utl_Dbws.QName;
lPortQName              Utl_Dbws.QName;
lStrResp                Varchar2(32767);
lService                Utl_Dbws.Service;
lServiceQName           Utl_Dbws.QName;
lXmlReq                 XMLType;
lXmlResp                XMLType;
Begin
    -- Set proxy if necessary
    -- Utl_Dbws.Set_Http_Proxy('192.168.161.123:3128');

    -- Create a service
    lServiceQName   := Utl_Dbws.To_QName('http://www.webserviceX.NET',
                                         'GetWeather');
    lService        := Utl_Dbws.Create_Service(lServiceQName);
    -- Set port
    lPortQName      := Utl_Dbws.To_QName('http://www.webserviceX.NET',
                                         'GlobalWeatherSoap');
    -- Set operation
    lOperationQName := Utl_Dbws.TO_QName('http://www.webserviceX.NET',
                                         'GetWeather');
    -- Create call
    lCall           := Utl_Dbws.Create_Call(lService      ,
                                            lPortQName    ,
                                            lOperationQName);
    Utl_Dbws.Set_Property(lCall, 'SOAPACTION_USE', 'TRUE');
    Utl_Dbws.Set_Property(lCall, 'SOAPACTION_URI',
                          'http://www.webserviceX.NET/GetWeather');
    -- Set endpoint address
    Utl_Dbws.Set_Target_Endpoint_Address(lCall,
                        'http://www.webserviceX.NET/globalweather.asmx');
    -- Set xml request
    lXmlReq := XmlType('<?xml version="1.0" encoding="utf-8"?>'
                    ||  '<GetWeather xmlns="http://www.webserviceX.NET">'
                    ||    '<CityName>'    || pCity    || '</CityName>'
                    ||    '<CountryName>' || pCountry || '</CountryName>'
                    ||  '</GetWeather>');
    -- Get xml response
    lXmlResp := Utl_Dbws.Invoke(lCall, lXmlReq);
    Utl_Dbws.Release_Call(lCall);
    Utl_Dbws.Release_Service(lService);
    lStrResp :=
    lXmlResp.Extract('/GetWeatherResponse/GetWeatherResult/child::node()',
                  'xmlns="http://www.webserviceX.NET"').GetStringVal();
    lStrResp := Dbms_Xmlgen.Convert(lStrResp, Dbms_Xmlgen.ENTITY_DECODE);
    lXmlResp := XMLType.CreateXml(lStrResp);
    lStrResp := XmlResp.Extract('/CurrentWeather/Temperature/child::node()',
                               'xmlns=""').GetStringVal();
   Return lStrResp;
End;
```

Figure 5: Calling a Web service using UTL_DBWS package

We have discussed so far the possibilities we have to access external, non-Oracle functionality from an Oracle Forms database application, using Web services. But how can we call from the outside a piece of functionality that is defined in an Oracle Forms database application?

4. Providing Web services

Exposing functionality to the outside world

We cannot expose functionality defined within Forms modules as Web services, but we can expose Oracle database objects. This means that if the functionality we want to expose as Web service resides in the Forms module, we need to migrate it to the database and expose it from there. Let us explore a bit more this idea of migrating the business logic.

4.1. Why moving Forms business logic to the DB?

Forms architecture is not making a clear separation between its visual and data components. In ADF, for instance, the Data Modeler (e.g. ADF Business Components (BC)) cannot refer objects defined in the Viewer layer. In Forms any program unit or trigger can refer visual components. Forms is offering therefore a greater flexibility in application development, while in ADF the not-so-flexible structure is compensated with greater degree of reuse: the functionality defined in ADF BC can be exposed as Web services and called from external applications. The only possibility we have to increase the degree of reusability in Forms applications is re-writing or separating the data manipulating business logic from the user interface one. After all, the business processes should not depend on the currently chosen user interface.

When moving the business logic from the Forms application to the database it is advisable to make it in a controlled way, so that the migrated code to be consolidated, for instance, by packing the functionally-related code in stored packages[4].

Moving the code to the database has a series of benefits: on one side, the migrated database code can be reused by other applications directly or exposed as Web services. On the other side, the Forms application will be simplified and therefore easier to migrate to other technologies like ADF[5] or APEX.

4.2. How to expose functionality stored in the DB?

The Oracle database as Web service provider

In a Service Oriented Architecture an Oracle database can take not only the Web service client role, as we have seen before, but also the Web service provider role. This way its interoperability with other applications has increased dramatically in the latest Oracle versions because we have now access to Oracle data and functionality from any environment that is able to call Web services.

What exactly can we expose? From the list of Oracle components that can be exposed as Web services: predefined SQL, XQuery or DML statements, Pl/SQL and Java stored procedures and Advanced Queues, we will focus now on PL/SQL stored procedures. The PL/SQL stored procedures like functions, procedures or packages are the ones that hold the core of our application business logic, and also the business logic migrated the from Forms applications to the database.

---

[4] PITSS.CON Application Engineering AE, A. Gaede, 2009,
http://pitss.com/fileadmin/pitss/images/de/White_Papers/2009-07_PITSS_White_Paper_AE_-_EN.pdf

[5] From Oracle Forms to Oracle ADF and J2EE, M. Serban, B. Us, 2009
http://www.pitss.de/fileadmin/pitss/images/de/White_Papers/2010-11_PITSS_White_Paper_ADF_EN.pdf

As depicted in Figure 6, we have three possibilities to expose these structures:

- **PL/SQL Web services** - represent the first and maybe the mostly used solution to expose stored procedures and functions as Web services. They need to be deployed on a Web server like WebLogic.
- **BPEL processes** - usually used to orchestrate complex processes; can also be used to expose stored procedures and functions. They need to be deployed on a Web server like WebLogic that contains a BPEL runtime.
- **Native XML DB Web services** – first released with Oracle 11g database, they represent the newest solution to expose stored procedures and functions as Web services. These Web services need no coding or Web server for deployment.

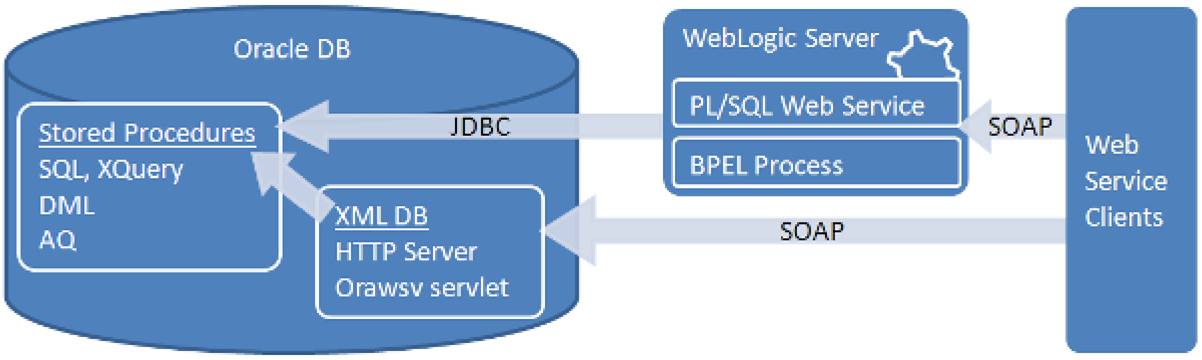


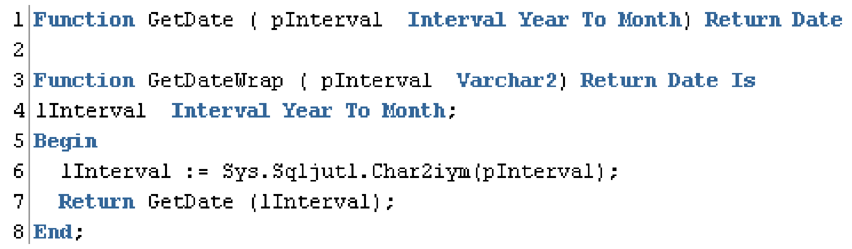
Figure 6: Oracle database as Web service provider

4.2.1. PL/SQL Web services

The PL/SQL Web services are usually assembled around one or more Java classes using JDBC to establish an Oracle database connection and call database stored program units.

Again, the question is: what exactly can we expose this way? Well, we can expose standalone procedures and functions and also procedures and functions defined in database packages. There are, however, some restrictions given by the Oracle JDBC drivers: not all SQL and PL/SQL types defined in the Oracle database are supported. We cannot expose, therefore, functions and procedures using as arguments SQL data types like **`Interval day to second`**, **`Interval year to month`**, **`Timestamp with local time zone`**, **`Timestamp with time zone`** or PL/SQL data types like **`PL/SQL boolean`**, **`PL/SQL record`** or **`PL/SQL index by table`**.

We have alternative solutions for these limitations: PL/SQL wrappers for these stored program units. We can define these wrappers to use only equivalent SQL data types supported by the JDBC drivers and then exposing them later as Web services. Figure 7 offers an example of equivalent data type for **`Interval year to month`**.



```
Function GetDate ( pInterval  Interval Year To Month) Return Date

Function GetDateWrap ( pInterval  Varchar2) Return Date Is
lInterval  Interval Year To Month;
Begin
  lInterval := Sys.Sqljutl.Char2iym(pInterval);
  Return GetDate (lInterval);
End;
```

Figure 7: PL/SQL wrapper defined for the GetDate function

The below data type mapping table contains equivalents for the above-listed unsupported data types. Here we may note that for the **`PL/SQL record`** or **`PL/SQL index by table`** we need to additionally create corresponding object or collection data types.

Data types not supported by JDBC	Equivalent structures
`Interval day to second`	Vachar2
`Interval year to month`	Vachar2
`Timestamp with local time zone`	Vachar2
`Timestamp with time zone`	Vachar2
`PL/SQL boolean`	Integer
`PL/SQL record`	Object Type (with equivalent structure)
`PL/SQL index by table`	Collection Type (with equivalent structure)

The manual creation of a PL/SQL Web service is a complicated process that requires a wide palette of skills. We have powerful wizards available that can ease our work, like the ones provided by JDeveloper 11g or PITSS.CON. The automatisation process may not cover all the cases, but using a wizard and adjusting the results is still quicker and easier than a full manual development. Typical limitations are:

- **Packaged overloaded units -** This limitation is specific to the WSDL document that prevents us to publish different operations sharing a common name. The solution is generating and publishing as Web service of a new, unique interface for the overloaded unit.
- **Standalone units –** JDeveloper wizard will not allow us to expose simple functions or procedures. If we really need to expose such units, PITSS.CON 8 is offering a powerful assistant covering these situations.
- **Unsupported data types like the above-listed + `binary_float`, `binary_double`, `nclob` -** The possible solution would be to use wrappers, as detailed above

The last step that needs to be performed is to deploy the PL/SQL Web services on a Web server. Here we can choose between Oracle WebLogic, Jboss, Tomcat or WebSphere server. The deployment process can be performed in many ways: from JDeveloper, using an Ant task or, if we decide for WebLogic server, from the WebLogic console.

4.2.2. BPEL processes

BPEL is an XML-based language designed to assist the generation of complex processes by orchestrating multiple Web services. Deployed on a Web server like Oracle WebLogic, or any other server that include a BPEL runtime, the BPEL processes can themselves be exposed as Web services.

To construct a BPEL process we can use JDeveloper BPEL Editor. Working with BPEL Editor is as easy as drag-and-drop, and does not presume any Java or PL/SQL skills.

We will not go now in too much detail, but will focus on BPEL capability to expose Oracle database functionalities as Web services. The key component that makes this possible is the BPEL DB adapter. This adapter is predefined by Oracle for accessing stored and standalone procedures or functions, performing DML or selects on tables, polling for new or changed records in a table or executing SQL statements. Let us see some of the capabilities or limitations of the DB Adapter:

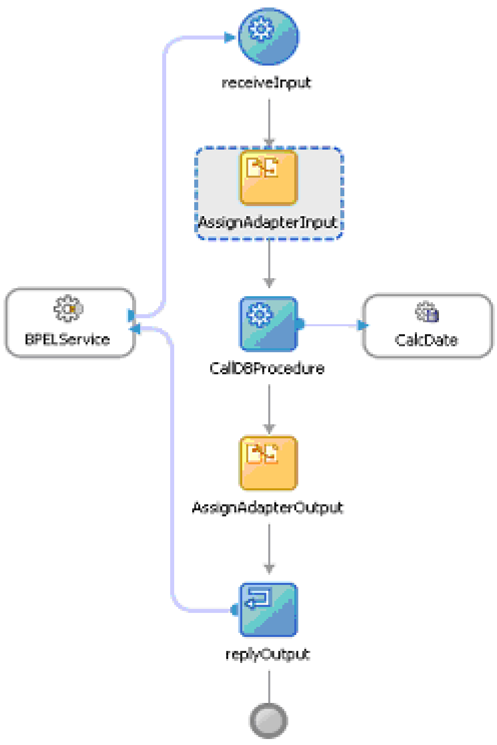


Figure 8: Example of a synchronous BPEL process able to expose a stored or standalone function

- **Overloaded**: It allows exposing overloaded functions or procedures.
- **Data types**: Does not support data types like `rowid`, `urowid`, `interval`, `timestamp with (local) time zone`, `bfile`. Like for the PL/SQL Web services, it can generate and expose wrapper functions or procedures where the unsupported data types are replaced with Varchar2. For the `PL/SQL record`, `PL/SQL index by table` and `PL/SQL Boolean` it generates and creates database wrappers and additional object or table types.
- The DB Adapter can be configured for only one stored standalone function or procedure. This is a serious limitation if we want to expose a whole database package, for instance. In such a case we would need to create DB adapters for each function or procedure in the package. If we have in the package several program units using unsupported types, then each corresponding DB adapter will create separate wrappers or additional object/ table types. This would lead to name conflicts and we would need to modify the created namespaces in order to eliminate these conflicts.

4.2.3. Native DB Web services

Native database Web services feature is an enhancement of Oracle XML DB, available starting with Oracle11g. With this feature Oracle eliminates the effort to assemble a Web service and more than that, the need of a deployment Web server. The Oracle database is here taking the Web server role, using its embedded HTTP server, part of Oracle XML DB feature.

By activating the native database Web services feature, the database procedures and functions, defined in a package or standalone, are automatically exposed as Web services. In addition, the native database Web services feature includes a Web service that can be used by the Web client applications to dynamically execute SQL statements and XQuery expressions.

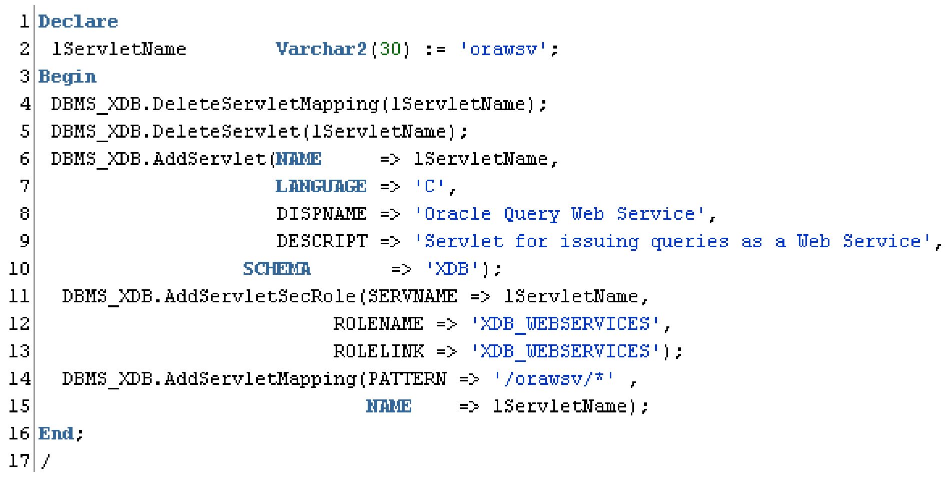
Activating the native DB Web services feature

For security reason, the native DB Web service feature is not activated in the Oracle database. To configure the Oracle XML DB to enable this feature we need to activate the Oracle XML DB HTTP server and configure the '`orawsv`' servlet in the `xdbconfig.xml` file.

Activating the XML DB HTTP server is done by setting its port number to a value that is bigger than 0. We can simply check and set the port number of the XML DB HTTP server by using the `DBMS_XDB.GetHttpPort` or `.SetHttpPort` functionality, like in the following command line:

```
SQL> EXEC dbms_xdb.sethttpport(8080);
```

The same `DBMS_XDB` package can be used to add the '`orawsv`' servlet configuration in the `xdbconfig.xml` Oracle configuration file, in a script like the one listed in Figure 9.



```
Declare
 lServletName        Varchar2(30) := 'orawsv';
Begin
 DBMS_XDB.DeleteServletMapping(lServletName);
 DBMS_XDB.DeleteServlet(lServletName);
 DBMS_XDB.AddServlet(NAME      => lServletName,
                     LANGUAGE => 'C',
                     DISPNAME => 'Oracle Query Web Service',
                     DESCRIPT => 'Servlet for issuing queries as a Web Service',
                  SCHEMA       => 'XDB');
  DBMS_XDB.AddServletSecRole(SERVNAME => lServletName,
                           ROLENAME => 'XDB_WEBSERVICES',
                           ROLELINK => 'XDB_WEBSERVICES');
  DBMS_XDB.AddServletMapping(PATTERN => '/orawsv/*' ,
                              NAME     => lServletName);
End;
/
```

Figure 9: Script for adding the `'orawsv'` servlet configuration to the XML DB configuration file (run from the SQL command line connected as SYS user)

Enabling native Web services for a DB user

Although the native database Web services feature is now enabled, the database users will need additional rights in order to be able to expose program units as native Web services. There are 3 roles that control the access to the native database Web services:

- **`XDB_WEBSERVICES`** – required role, enabling the use of native Web services over HTTPS
- **`XDB_WEBSERVICES_OVER_HTTP`** – enabling the use of native Web services over HTTP
- **`XDB_WEBSERVICES_WITH_PUBLIC`** – enabling access to the PUBLIC database objects.

Locating and accessing native DB Web services

After activating the native database Web services feature and granting the appropriate roles to the database user, the next step will be to identify the Web services location, i.e. the URLs of the native Web services WSDL documents. The WSDL document gives us not only the location of the Web service but also its operations and corresponding messages. We may note here the Oracle XML DB supports SOAP1.1 and the Web service client will need to use the HTTP POST method when submitting SOAP request to the native database Web services.

The native database Web services feature includes only one Web service that can be used to execute SQL statements or XQuery expressions in database. Its WSDL document URL is located at an address like http://host:port/orawsv?wsdl, where the host is the database host and the port is the port number used by the XML DB HTTP server.

For PL/SQL program units we have a diferrent situation. Each procedure or function, standalone or defined in a package, has its own dynamic Web service. The WSDL documents for these Web services are located at http://host:port/orawsv/DBSCHEMA/FUNCTION_or_PROCEDURE?wsdl, for standalone functions and procedures, and at http://host:port/orawsv/DBSCHEMA/PACKAGE/FUNCTION_or_PROCEDURE?wsdl for packaged functions and procedures. In addition, there are native Web services defined for database packages. Such a Web service exposes all procedures and function defined in the associated package and the URL of its WSDL document is http://host:port/orawsv/DBSCHEMA/PACKAGE?wsdl.
Here we have to pay attention that the schema, package, function or procedure name used in URLs must be written in uppercase, or otherwise the browser will return an error.

After we saw the advantages of the native DB Web services what can we say about their limitations? Some typical issues we can meet are:

- **Overloading**: Overloaded functions or procedures can be accessed only through the native Web service corresponding to the package they belong.
- **Supported data types**: The procedures and functions having parameters defined as PL/SQL record, PL/SQL indexed tables or database collection cannot be exposed as native Web services.

5. Conclusion

We have discussed the possibilities we have to better integrate Oracle Forms applications in the modern Web development world. As we are constantly confronted with new, modern technology choices for developing database applications, our task of choosing among the best of them is more challenging than ever. Take a look at Oracle conferences: the agendas are filled with new subjects, like ADF, APEX, BI Publisher and so on. And it is right so, as a new application development era has long begun: the Web development era. But, while we assess the strategies on the best way to go, we are confronted with a very down-to-earth, important and urgent matter: maintaining the current systems we have. Making them sustain the current businesses while using the currently available resources.

Lots of companies owning legacy Forms applications still choose a 'wait-and-see' approach, upgrading, for the moment, the Forms applications to the 11g stack. This way they get the benefits of the 11g capabilities, stay on supported platforms, while gaining time to assess the new technologies, to learn the modern development languages and architectures, and allowing the new technologies to get mature.

But whether we choose to migrate to new technologies or stay, for the moment, with Forms, a good strategy is to move as much as possible of the Forms application business logic to the database. This way we will later be able to migrate much easier to new technologies. The code in the database and the DB Web services will be easily reused in any technology we would later choose.

Implementing Web services and orchestrating them in a modern architecture can offer excellent results and can be obtained with a minimum investment, reflecting the core of the service-oriented (SOA) principles: reuse, interoperability and standards-compliance, helping our database applications to be more flexible and respond more quickly to the changing business requirements.

About PITSS

PITSS is the leading supplier of fully integrated solutions for effective management of Oracle Forms applications. The innovative PITSS.CON software helps its customers to analyse, migrate, upgrade and maintain their Oracle Forms applications in its entirety. PITSS thus opens an evolutionary path for the migration of Oracle Forms applications to a Service Oriented Architecture (SOA). PITSS.CON has earned a reputation through its high level of automation and performance. Migration and development projects are run rapidly, economically and reliably within shortest possible time frames. With PITSS.CON, companies achieve an average cost saving of 30% for regular development projects and up to 90% for upgrade projects. PITSS is an Oracle Certified Advantage Partner and has customers in Europe, USA and Asia.

Implementing Web Services in Oracle Database Applications

February 2011

Author: Florin Serban
Co-Authors: Magdalena Serban, Andreas Gaede

PITSS in Europe
Germany
+49-711-728.752.00
info@pitss.com
www.pitss.com

PITSS in Americas
USA
248.740.0935
info@pitssamerica.com
www.pitssamerica.com