Warehouse Builder 11*g*

ORACLE WAREHOUSE BUILDER: WEB SERVICES AND SOA-READY

May 2008

ORACLE

Oracle Warehouse Builder: Web Services and SOA Ready

INTRODUCTION: DATA INTEGRATION WITH OWB

Data integration problems, at least in their general outline, are by now familiar to all of us. The details vary, but at root, the problem, time and again, is "how do I get data from various sources, transform it, and make it available to various consumers?"

One classic version of the problem is the data warehousing problem: moving data from sources, to a staging area, then integrating it, and pushing it out to an array of targets, such as a data warehouse, data marts, master data management systems, and so on. Other scenarios might include, a simpler one-time movement of data, federation of data sources, data migrations where a gradual transition from an old system to a new system is required, and so on.

But there are two technical problems that must always be answered when tackling the integration question: how do I actually gain access to data in different sources, and publish the results of data integration to make them accessible for consumers? And how do I orchestrate the activities involved in the data integration process?

SOA: New Technologies, Old Questions

The SOA (Service-Oriented Architecture) approach, reframing applications as collections of loosely-coupled software services communicating using open standards like SOAP and HTTP, may be the new model for application architecture in the age of the Web, but it still presents these familiar fundamental problems, if in new guises. Services may be built and combined to make applications while being more agnostic about the technologies to build each component, and WSDL as the basis for describing their interactions, but the basic needs in integration scenarios are not so different from before. Any modern piece of enterprise software must be equipped to support this new model of interaction, just as earlier models like client-server were and continue to be supported.

Oracle Database Support for Web Services

Fortunately, Oracle Database and Oracle Warehouse Builder have been ready to participate in SOA-based solutions for several releases. Oracle Database support for Web Services has been present since the Database 9i release, and was

significantly enhanced in Database 10g and 11g. Database operations and data retrieval can be executed through Web Services mechanisms.

In conjunction with Oracle Application Server Web Services Framework, Oracle Database allows instant interoperability and consistent Web services development and deployment. External Web Services can be included as part of a SQL query or database batch processing. Oracle Database 10g allows you to define the functionality behind a web service using PL/SQL (packages, procedures, functions, and triggers), SQL queries, SQL DML statements, and Java stored procedures. Any PL/SQL package can be published as a Web Service.

Supporting a web services API opens up mappings and process flows to any J2EE (tm) or .NET application. As a result, you can take advantage of your investment in Oracle SQL and PL/SQL code and development tools and skills to create applications that are accessible via SOAP from applications built on any modern platform.

Oracle JPublisher Utility

The Oracle JPublisher utility facilitates setting up web service and SOA access for the Oracle database (both calling and consuming web services). JPublisher is generates Java classes to represent database entities, such as SQL objects and PL/SQL packages, in a Java client program. It also provides support for publishing from SQL, PL/SQL, and server-side Java to Web services and enabling the invocation of external Web services from inside the database.

The use of JPublisher in publishing and consuming Web services from the Oracle Database is described where needed in the examples in this document. Note, however, that some of the specific behaviors of JPublisher change from one version of the Oracle Database to the next. For more detailed information about JPublisher, consult the documentation for your specific version of the Oracle Database.

Oracle Warehouse Builder Support for SOA and Web Services

Oracle Warehouse Builder leverages the support for Web Services from the Oracle Database to provide a full range of SOA capabilities. The use cases supported as a result include:

- Consuming data sources exposed as Web Services in OWB process flows
- Exposing an OWB mapping as a Web service that transforms submitted data and returns the result to the caller
- Exposing an OWB mapping as a Web service that can be used to load data into any Warehouse Builder target
- Exposing an OWB process flow as a Web service for invocation from an outside business process orchestration tool such as Oracle BPEL

The sections of the paper that follow will describe how to perform each of these tasks, and the practical value of each case. Because we understand that many of our customers are still using OWB 10.2, the examples in this paper will focus on that version, although the methods of performing these tasks in OWB 11.1 are extremely similar.

EXAMPLE: CONSUMING WEB SERVICES IN OWB MAPPINGS

There are a number of ways to execute web services from within the database and therefore from with OWB-generated PL/SQL:

- Perform all the work using UTL_HTTP
- Use Jpublisher which does a lot of work to interface SQL to the web service
- Use the UTL_DBWS package that allows generic web service consumption. (Note that, depending upon your database version, you may have to download and install the DBWS utility to perform these tasks. You can find the appropriate version of DBWS on OTN.)

A detailed walk-through of calling a Web service from PL/SQL is available on the Oracle Technology Network web site, here:

http://www.oracle.com/technology/tech/java/oc4j/1003/how_to/how-to-ws-db-javacallout.html

To consume a web service in OWB, include the call-out in an OWB mapping, as shown in this example.

The example uses a Web service created for the 2006 World Cup. The example will take a table with a list of stadium names and retrieve the city name and capacity of each stadium from the web service. The table STADS is created as follows:

```
create table stads (name varchar2(256));
insert into stads values ('Olympiastadion');
insert into stads values ('Zentralstadion');
commit;
```

This example leverages the JPublisher code that generates object types, tables of objects and table functions from a WSDL file from the web service. At the time of publication, the WSDL file was available here:

```
http://www.dataaccess.nl/wk2006/footballpoolwebservice.wso?WSDL
```

Step 1: Create Proxies from WSDL

Use the jpub utility to generate table function proxies and publish PLSQL wrapper and proxy code for the WSDL into the schema.

```
jpub -user=<user>/<password> "-sysuser=sys/passwd"
      "-compile=true" "-proxyopts=tabfun" "-proxywsdl=
      http://host:port/footballpoolwebservice.wsdl"
```

where user and password are database credentials for the OWB location where the process flow will run. (You can also generate the code to be loaded later if

necessary, by specifying the "-proxyopts=tabfun,noload" option and omitting the user and sysuser credentials.)

The generated code includes SQL scripts with the type definitions, functions and table functions and also Java code that must be loaded into the database that binds the SQL, Java and Web Service worlds.

Step 2: Define PL/SQL Code to Call Web Service

The PL/SQL code that can call out to the Web service (independent of OWB) is:

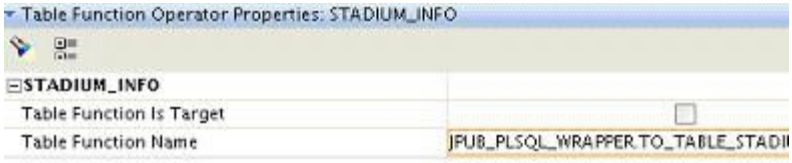
```
Select
    r.arg0 name, r.res.iSeatsCapacity capacity,
    r.res.sCityName_city
    from table(JPUB_PLSQL_WRAPPER.to_table_stadiuminfo(
        cursor(select name from stads))) r;
```

Step 3: Import Metadata into OWB

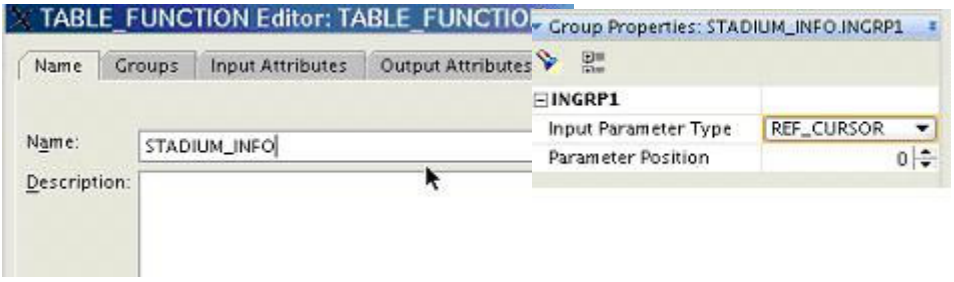
Import the metadata for the generated package into OWB (or at a minimum, the types and table types). Table function support is loosely coupled in OWB 10.2 so there is not a great value in reverse engineering the package generated by jpub if you need the table functions; you would still have to enter the table function names in the table operator. But since the types are certainly used, you should import all the types and table types. Also import the table STADS that has been created above. You will use this table in a mapping that processes the data from the web service.

Step 4: Define a Mapping with Table Function to Call the Web Service

Create a mapping, then add a table function operator to the mapping. Enter the name of one of the generated table functions in the operator's property panel. This is the function that will be called on the web service. For readability, you may want to change the name of the operator.



Creating a Table Function Operator to Call a JPUB Wrapper Function



Renaming the Table Function Operator

Add the input attribute for the function being used (this is taken from the dependent function), and the output attribute based on the type returned by the table function (you can assume OBJECT_VALUE here).

Step 5: Capture the Output of the Web Service in a Table

To capture the web service output:

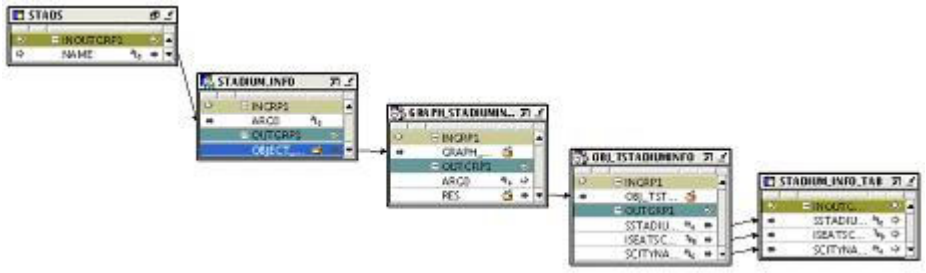
- Add an EXPAND operator for the type returned by the table function and another EXPAND operator for the RES value (which is of type OBJ_TSTADIUMINFO). This will expand into the structure for each row returned by the web service.
- Add a table operator, and create a new table to be bound to it. All records returned from the web service will be written to this table.
- Map from the output group of the expand operator to the new table operator's group.

Step 6: Define the Source Rows for Input to the Web Service

To set up the inputs for the Web Service:

- Define the input to be a REF_CURSOR, based on the STADS source table.

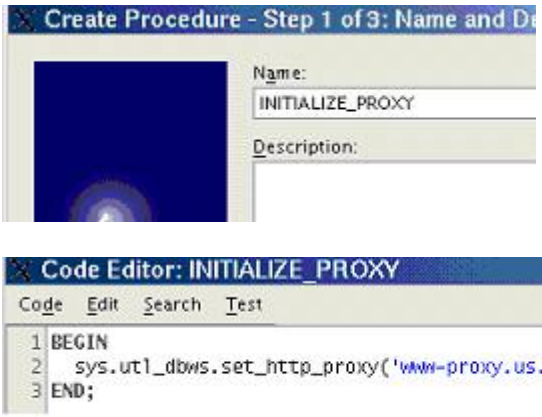
- Add the STADS source table and map the NAME attribute into the table function's input.



Web Service Mapping with input and output defined

Step 7: Set HTTP Proxy (if needed)

- If you have to set a web proxy, for example because you are running behind a corporate firewall, then create a pre-mapping process in the map that uses uses the procedure INITIALIZE_PROXY to configure an HTTP proxy.



Creating the INITIALIZE_PROXY Pre-Mapping Process

Executing the Mapping

After executing the mapping we see the following content in the target:

Results in the target table, shown in the Data Viewer

EXAMPLE: EXPOSING NAME AND ADDRESS CLEANSING MAPPING AS A WEB SERVICE

This section describes how to expose a mapping as a web service. The solution described leverages Warehouse Builder's data quality (in this case, address cleansing) capabilities, but of course any OWB data transformation that can be represented as a mapping can be invoked in a similar fashion.

Two implementations will be discussed: one single-record implementation and one multiple-record implementation. In real life cases where a mapping will be used heavily as a web service, you will probably want to optimize the implementation for multi-request situations and/or specific performance requirements.

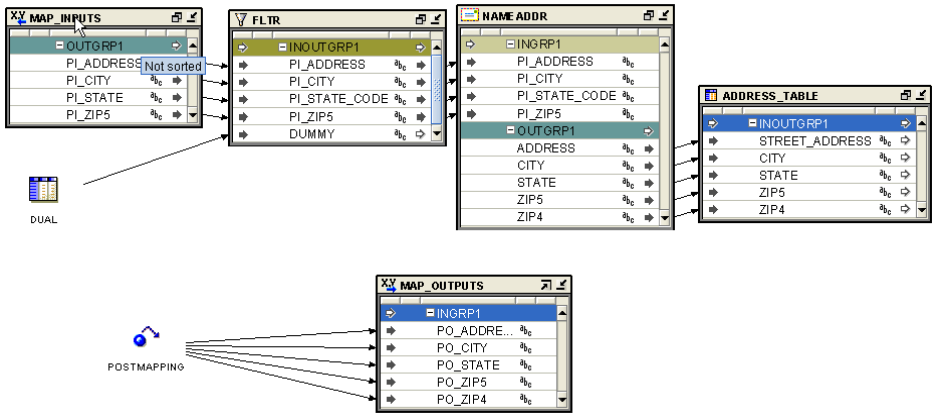
The example shown here describes a mapping that accepts a US address as input variables and returns a standardized, cleansed result including zip+4 (US standard) as output.

This discussion assumes that the mapping is already defined, and focuses on the steps needed to publish and use the mapping as a web service.

This same technique can be used with any version greater than 9.0.4.10.

Step 1: Review The Cleansing Mapping

The figure below shows a screenshot of the mapping. We will examine each of the operators in this map to see how they fit together to make a procedure that can be called repeatedly.



The Name/Address Cleansing Mapping

Filter operator and the mapping inputs

In the Warehouse Builder mapping paradigm, a mapping always has to process at least one input record from a data source. A map to be exposed as a web service must take input parameters (and return a record with its outputs). To turn the input parameter values into a record, a Filter operator is used, taking a select from the DUAL database table (SELECT * FROM DUAL WHERE 1=1), and adding the parameters as extra columns by adding them to the filter's INOUTGROUP. The filter is being used to combine the attribute from DUAL and the parameters. The filter condition is 1=1, so the record is always passed through.

Name and Address Operator

Into the name and address operator go only the address related input attributes. Name and address outputs are individual items to be loaded into the ADDRESS_TABLE.

ADDRESS Table Operator

The loading type on ADDRESS_TABLE is TRUNCATE/INSERT. Each time the mapping is run, the table is truncated and then the cleansed row from the mapping is inserted.

Post-Mapping Operation to Publish Transformed Rows

A post mapping procedure reads the transformed row from the ADDRESS_TABLE and copies the results to the mapping output attributes.

Invoking the mapping from PL/SQL

Warehouse Builder provides a generic framework for running mappings from PL/SQL. You can find sample code for invoking a mapping in the file owb_home/owb/rtp/sql/sqlplus_exec_template.sql. In this instance, a somewhat extended version of the template is being used in order to invoke the mapping and pass in the input parameters.

For this business case, the target user was used as the user who invokes the mappings. If implementing this yourself, you must make sure the grants are correct in order to be able to run the mapping through PL/SQL:

- Grant execute on RUN_OWB_CODE to the user you want to use to invoke the mapping, and in the schema you use to invoke the mapping, create a local synonym RUN_OWB_CODE to point to <runtime repository>.RUN_OWB_CODE.
- Grant execute on <runtime repository>.WB_RT_API_EXEC to the user you use to invoke the mapping.
- Grant create procedure to <access user>. This is needed in order to create a package that calls the RUN_OWB_CODE procedure.
- Grant the 4 runtime roles to the user in order to promote the target user to a runtime access user: WB_A_<runtime repository>, WB_D_<runtime repository>, WB_R_<runtime repository> and WB_U_<runtime repository>. The target user acts as a runtime access user in this case.

Because the MAP_NA mapping has input and output parameters, you must define a (fairly straightforward) specific wrapper that calls the generic wrapper. The code for the wrapper package/procedure is included here:

```
CREATE OR REPLACE PACKAGE INVOKE_MAP_NA IS
  PROCEDURE  CALL_MAPPING ( P_IN_ADDRESS IN VARCHAR2
                          , P_IN_CITY IN VARCHAR2
                          , P_IN_STATE IN VARCHAR2
                          , P_IN_ZIP5 IN VARCHAR2
                          , P_OUT_ADDRESS OUT NOCOPY VARCHAR2
                          , P_OUT_CITY OUT NOCOPY VARCHAR2
                          , P_OUT_STATE OUT NOCOPY VARCHAR2
                          , P_OUT_ZIP5 OUT NOCOPY VARCHAR2
                          , P_OUT_ZIP4 OUT NOCOPY VARCHAR2
                          )  ;
END INVOKE_MAP_NA;
/

CREATE OR REPLACE PACKAGE BODY INVOKE_MAP_NA AS

  PROCEDURE  CALL_MAPPING ( P_IN_ADDRESS IN VARCHAR2
                          , P_IN_CITY IN VARCHAR2
                          , P_IN_STATE IN VARCHAR2
                          , P_IN_ZIP5 IN VARCHAR2
                          , P_OUT_ADDRESS OUT NOCOPY VARCHAR2
```

```
                          , P_OUT_CITY OUT NOCOPY VARCHAR2
                          , P_OUT_STATE OUT NOCOPY VARCHAR2
                          , P_OUT_ZIP5 OUT NOCOPY VARCHAR2
                          , P_OUT_ZIP4 OUT NOCOPY VARCHAR2
                          ) IS
    l_result number ;
    l_audit_id number ;
  begin
    rtr.run_owb_code( l_result
    , l_audit_id
    , 'RTR'
    , 'WS_TARGET_LOC'
    , 'PLSQL'
    , 'MAP_NA'
    , '","'
    , 'PI_ADDRESS='
      || p_in_address
      || ',PI_CITY='
      || p_in_city
      || ',PI_STATE='
      || p_in_state
      || ',PI_ZIP5='
      || p_in_zip5
    ) ;
    if l_result = 1
    then
      dbms_output.put_line('Mapping execution was successful') ;
      p_out_address :=
rtr.wb_rt_api_exec.get_output_parameter(l_audit_id,'PO_ADDRESS') ;
      p_out_city :=
rtr.wb_rt_api_exec.get_output_parameter(l_audit_id,'PO_CITY') ;
      p_out_state :=
rtr.wb_rt_api_exec.get_output_parameter(l_audit_id,'PO_STATE') ;
      p_out_zip5 :=
rtr.wb_rt_api_exec.get_output_parameter(l_audit_id,'PO_ZIP5') ;
      p_out_zip4 :=
rtr.wb_rt_api_exec.get_output_parameter(l_audit_id,'PO_ZIP4') ;
    elsif l_result = 2
    then
      dbms_output.put_line('Mapping execution with warning') ;
    else
      dbms_output.put_line('Mapping execution failed') ;
    end if ;
    execute immediate 'alter session set current_schema = rt' ;
  end ;


END INVOKE_MAP_NA;
/
```

Notice that in this case schema RT is being used to invoke the mapping. The wrapper RUN_OWB_CODE that implements the sqlplus_exec_template.sql performs an 'alter session set current_schema = <runtime repository>'. The EXECUTE IMMEDIATE statement at the end of the transformation changes the current schema back to the original schema, so that all procedures etc. can be found based on the current schema again. If you want to use this example for your business case, then you may have to update this value to include the user schema you use.

Warehouse Builder Deployment

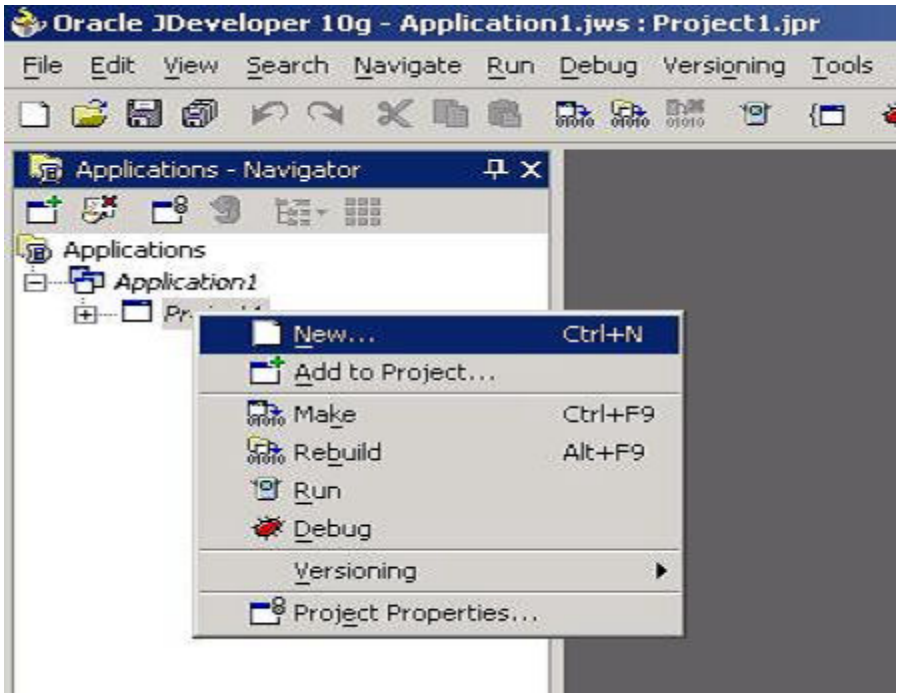
All code must be deployed to the database. This includes the mapping and the necessary transformations. If you use your target schema to invoke the mapping, then all code can be deployed from the MDL file.

Publishing the package as a web service

You can use JDeveloper in order to define a web service that calls PL/SQL in the database.

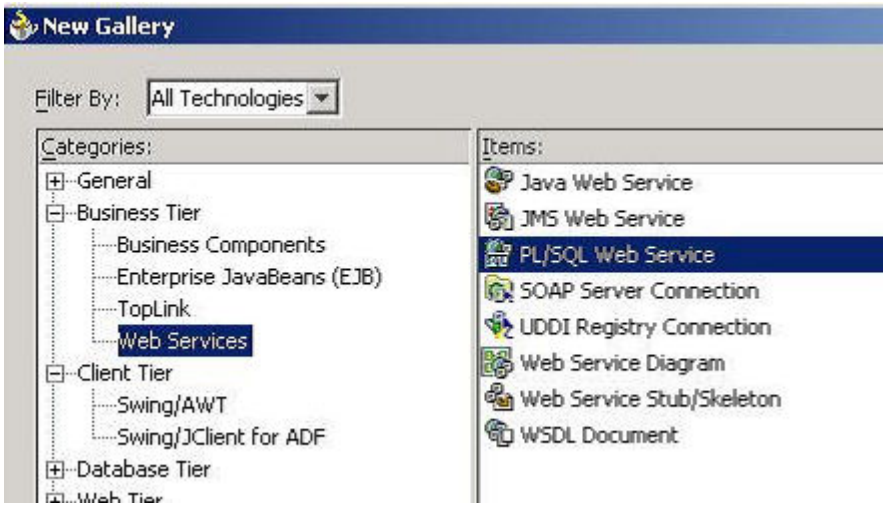
Note: The procedure described here uses the 10g version of JDeveloper. Earlier versions do not support procedures with output parameters as web services.

Once you have defined a new (empty) Jdeveloper project, you can use the New option from the right-mouse popup menu on the project to define a new web service (see below).

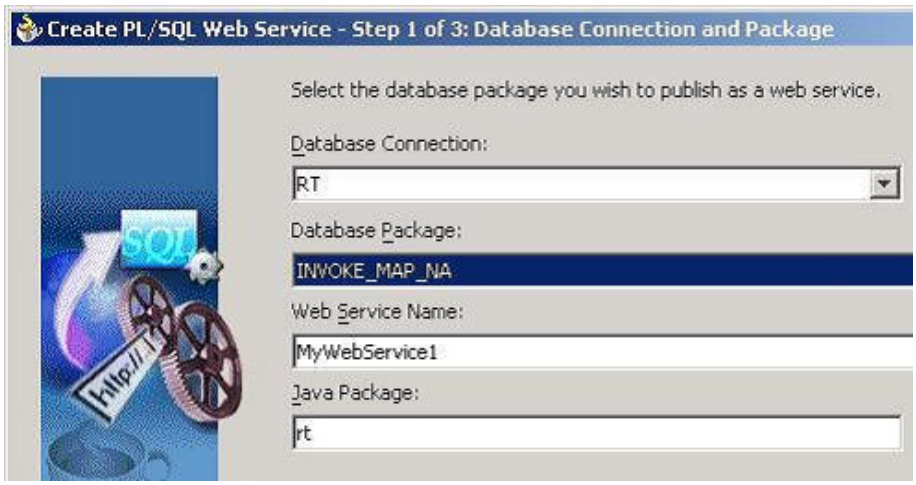


JDeveloper with a new empty project.

Use the PL/SQL Web Service under the Business Tier in Jdeveloper.

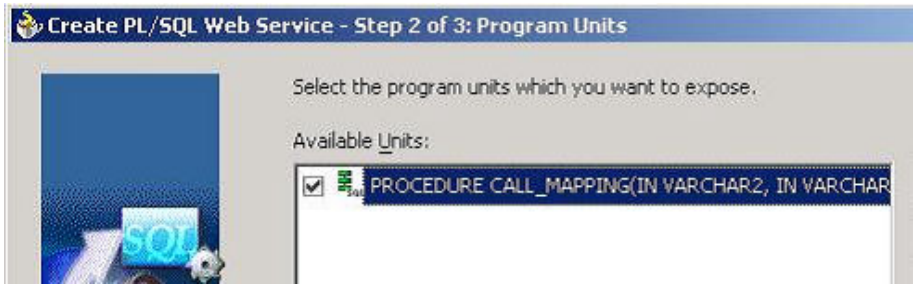


Go through the wizard to point to the database schema, and select the procedure in the package and define the web service.



Select the INVOKE_MAP_NA package to be published as a web service, as shown above.

In Step 2, select the CALL_MAPPING procedure to be exposed as a web service.

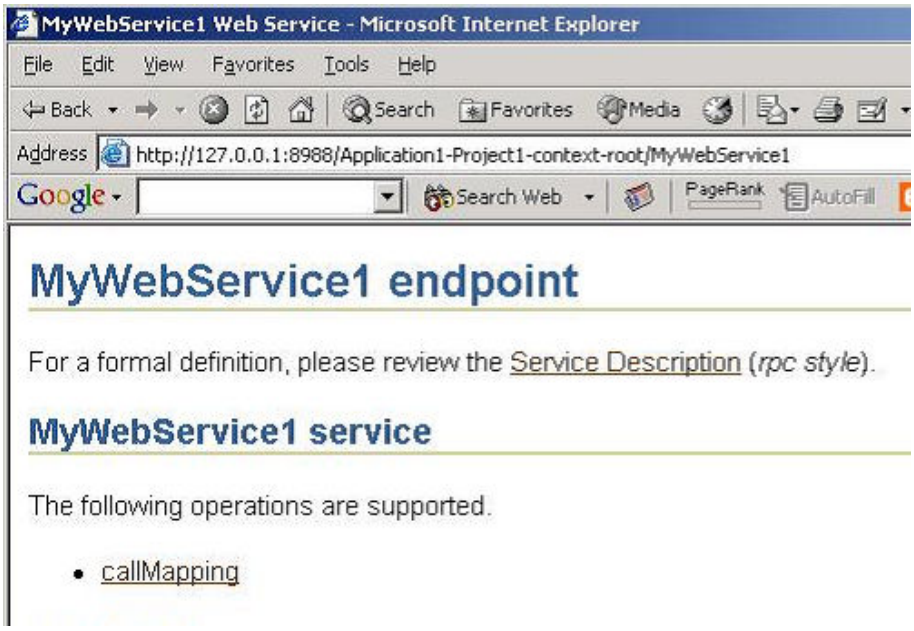


Click Next.

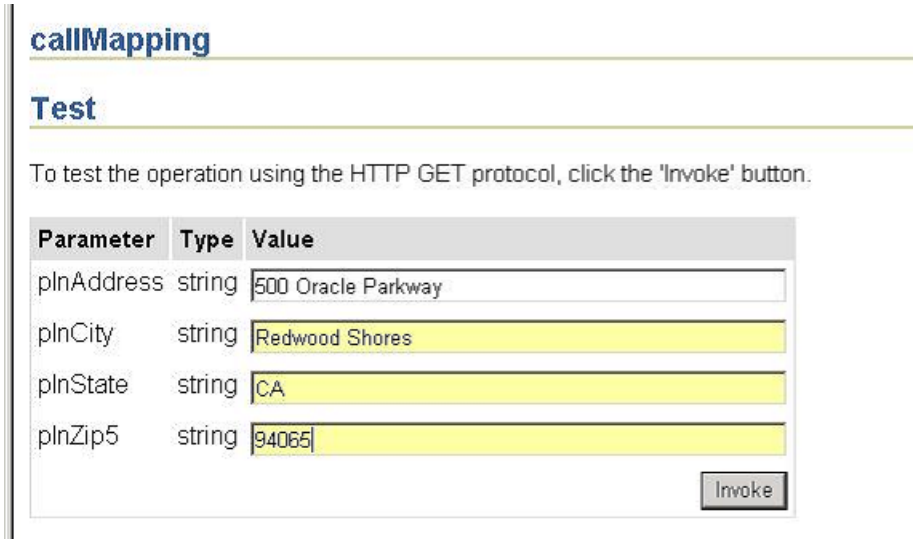
Once you complete the wizard JDeveloper will generate all code you need in order to publish the web service and deploy it to a target OC4J instance. Once you have the code, use the right-mouse button popup menu on MyWebService1 and select Run. JDeveloper will then invoke its own OC4J instance and host the web service. Notice the URL, specifically the listener port, in the bottom right panel. This is the URL to the web service.

Invoking the web service

Finally, you can copy and paste the URL from the Embedded OC4J Server panel into your web browser in order to connect to the web service. This link provides you the link to the web service definition.



To invoke the Web service, click the callMapping link. A form appears, where you specify the values to pass to the Web service.



Fill in the form with an address, and click the Invoke button. This uses the Web Service to invoke the mapping. The mapping performs address verification and validation using Warehouse Builder's name and address cleansing operator and returns the result in a SOAP message.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  - <SOAP-ENV:Body>
    - <ns1:callMappingResponse xmlns:ns1="http://rt/MyWebService1.wsdl" SOAP-
        ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      - <return xmlns:ns2="http://rt/MyWebService1.xsd"
          xsi:type="ns2:rt_MyWebService1Impl_callMapping_Out">
          <poutaddressOut xsi:type="xsd:string">500 ORACLE PKWY</poutaddressOut>
          <poutcityOut xsi:type="xsd:string">REDWOOD CITY</poutcityOut>
          <poutstateOut xsi:type="xsd:string">CA</poutstateOut>
          <poutzip4Out xsi:type="xsd:string">1677</poutzip4Out>
          <poutzip5Out xsi:type="xsd:string">94065</poutzip5Out>
        </return>
      </ns1:callMappingResponse>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

Notice that the address values are standardized to uppercase and Redwood Shores is being replaced by REDWOOD CITY. Also, the ZIP+4 is being returned.

Because the mapping runs in Warehouse Builder's framework, it populates the audit details like you would expect from any other mapping. You can view the results in the audit browser.

This end-to-end example shows how you can interact with a Warehouse Builder mapping in realtime, running as a web service.

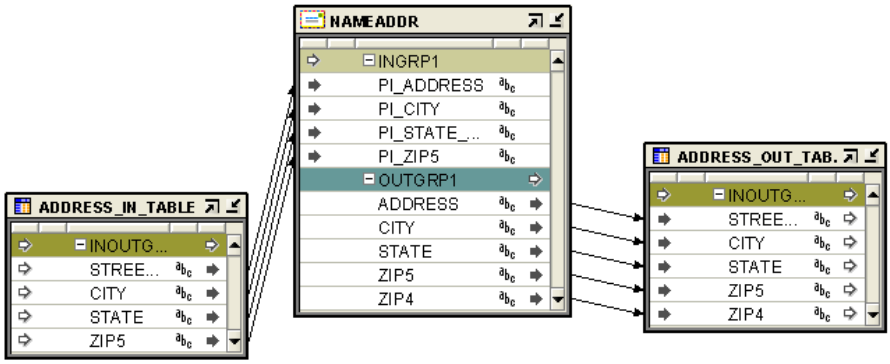
This example can be extended to execute any mapping or process flow generated from Warehouse Builder. Of course, not all mappings or process flows return data values. What they do return is the status of execution: successful, with warnings, or

error. You could use this example to define a generic framework to execute any mapping or process flow that was generated from Warehouse Builder and return the status after the execution.

EXPOSING A WEB SERVICE TO PROCESS MULTIPLE RECORDS

Even though the single-record business case shows the proof of concept and can satisfy many requirements, it has restricted scalability. A bulk interface, that accepts multiple records and returns multiple records, is more scalable and can satisfy more requirements accordingly. This section describes the multi-record case. Because the principle of this second example is the same as the previous example, certain steps will be described very briefly.

Figure 12 below shows the MAP_NA_BULK mapping that is used to implement the web service that processes multiple records.



The Bulk Name and Address Cleansing Mapping

Notice that, unlike the single-record mapping, this one has no input and output parameters. Instead, this mapping is reads its inputs from a source table and inserts the outputs into a target table. In this case, the PL/SQL code that calls the wrapper has to do the translation of input into records in the source table.

Invoking the mapping through PL/SQL

In order to make the translation between multiple input values and records in a table, to be called by a web service, you have to use database object types. Please refer to script create_na_bulk_PLSQL.sql for the full code to implement the business case.

```
create type address_in_type is object
( address varchar2(50)
, city varchar2(50)
, state varchar2(20)
, zip5 varchar2(5)
) ;
/
create type address_out_type is object
( address varchar2(50)
, city varchar2(50)
, state varchar2(20)
, zip5 varchar2(5)
, zip4 varchar2(4)
```

```
) ;
/
create type addresses_in is table of address_in_type
/
create type addresses_out is table of address_out_type
/
```

Next, the PL/SQL wrapper that calls the mapping makes the translation between the object type and the records in the table. In this case, a database function in a package is being used to invoke the mapping. The function returns an object of type ADDRESSES_OUT, which is a table of ADDRESS_OUT_TYPE.

This package is included in script create_na_bulk_PLSQL.sql.

```
create or replace package na_bulk is
  function invoke (p_address in addresses_in) return addresses_out ;
end na_bulk ;
/
create or replace package body na_bulk is
  function invoke (p_address in addresses_in) return addresses_out
  is
    l_result number ;
    l_audit_id number ;
    cursor c_addresses_out is
    select address_out_type(street_address
    , city, state, zip5, zip4
    ) my_address
    from address_out_table ;
    l_addresses_out addresses_out ;
  begin
    execute immediate 'truncate table address_in_table' ;
    for i in 1..p_address.count
    loop
      insert into address_in_table
      (street_address, city, state, zip5)
      values
      ( p_address(i).address
      , p_address(i).city
      , p_address(i).state
      , p_address(i).zip5
      ) ;
    end loop ;
    commit ;
    run_owb_code( l_result
    , l_audit_id
    , 'RTR'
    , 'WS_TARGET_LOC'
    , 'PLSQL'
    , 'MAP_NA_BULK'
    ) ;
    open c_addresses_out ;
    fetch c_addresses_out
    bulk collect into l_addresses_out ;
    close c_addresses_out ;
    execute immediate 'alter session set current_schema =
target_schema' ;
    return l_addresses_out ;
  end ;
end na_bulk ;
/
```

Notice that the target user in this example is *target_schema*. If you are trying to recreate this example, you may need to change this in your environment to match your target schema.

Warehouse Builder Deployment

Deploy the mapping to the database, and run the script to create the PL/SQL definitions into the Warehouse Builder target schema.

Publishing the package as a web service

In a similar way as described in the first business case, you go through the wizards in JDeveloper in order to define and expose the web service. Please refer to the single-record case for the details and the steps to go through.

Invoking the multi-record cleansing web service

Once you have defined the web service, run it in JDeveloper. JDeveloper will again show the URL to retrieve the web service definition. Open this URL in the browser and invoke the web service. Then compare the ADDRESS_IN_TABLE and ADDRESS_OUT_TABLE contents using the data viewers. Note that the ADDRESS_IN_TABLE had a mis-spelling of San Francisco in one address, and referred to Redwood Shores in another. The cleansed output has corrected the spelling of San Francisco and standardized Redwood Shores to REDWOOD CITY.

Note: Even though not every Warehouse Builder mapping or process flow returns data (or has to return data), it will always return the resulting status after processing. The call to the web service would invoke the mapping or process flow and return the status to the application used to invoke the web service.

Deploying to Production Application Servers

The business cases so far have used the embedded OC4J engine in JDeveloper in order to host the web service. In a production environment, you would typically not do that, but deploy to an application server instead. An application server can be as extensive as Oracle's Application Server 10g, but it can also be just a standalone OC4J engine. This section describes how to deploy the web service to the standalone OC4J engine that ships with Warehouse Builder. Unless you use Oracle Application Server to host your design browser and runtime audit browser you would invoke the standalone OC4J engine in order to access the design browser or the runtime audit browser.

JDeveloper's wizard has already generated the deployment specification for the web services (the definition of the WAR file that is needed to deploy to an application server). All you need to do is define a connection to the application server, and use the deployment specification to deploy to the server. In JDeveloper 10g, you use the Connections tab to define a connection to any of the supported application servers. Once you have your application server connection defined and tested, you can deploy to it. Go back to the Applications tab, to the context of your JDeveloper project, and right click on the deployment specification in order to deploy to the application server. The deployment log in the right bottom panel

shows the URL that can be used to access the web service (suffix this URL with the respective web service).

LOADING DATA INTO A DATABASE TARGET VIA A WEB SERVICE

The mechanism for loading data into a target using a mapping exposed as a web service is straightforward, given the preceding examples. Simply make one of the targets of your mapping the table to be loaded with transformed data. In such a case, loading the passed data can be thought of as a side effect of the mapping, where the primary effect is to return some status information to the caller. Your mapping can use the mechanisms described in the previous sections to return success, warnings or error information to the caller.

SUMMARY

The preceding examples should demonstrate how you can integrate Warehouse Builder into SOA-based solutions. While these examples focused on data cleansing and either passing values into the web service or sourcing from Oracle tables, any data source or target known to OWB and any operators or process flow tasks could have been used.

Thus, the full data integration, transformation and data quality capabilities of OWB as expressed in mappings and process flows can be exposed for invocation as part of a larger distributed application, regardless of the underlying technologies used to implement various non-OWB components. Sources such as ERP applications and non-Oracle databases accessed via gateways or ODBC can incorporated into any SOA-based application. OWB is therefore a valuable element that can be incorporated into any solution that uses an Oracle database, even if it is implemented in the SOA style.

RESOURCES

- Oracle Warehouse Builder Blog: Publishing Process Flow as a Web Service http://blogs.oracle.com/warehousebuilder/newsItems/viewFullItem$168
- Oracle Database 10g Web Services on OTN: http://www.oracle.com/technology/tech/webservices/database.html
- Oracle Database 10g Web Services Sample Code: http://www.oracle.com/technology/sample_code/tech/java/jsp/dbwebservices.html
- Web Service Callout User Guide: Invoking Web services from PL/SQL with sample code http://www.oracle.com/technology/sample_code/tech/java/jsp/callout_users_guide.htm

Oracle Warehouse Builder: Web Services and SOA Ready
May 2008
Author: Antonio Romero

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com