An Oracle White Paper
February 2014

RESTful Web Services for the Oracle Database Cloud

Table of Contents

Introduction to RESTful Web Services

REST, the commonly used name for a set of architectural principals officially known as Representational State Transfer, has become a standard for defining Web services on the Web, based in large part on its simplicity when compared with two other widely used architectures, SOAP and WSDL-based services. REST is widely used by a variety of Web service providers, including Google, Facebook and Yahoo, as the way to expose their own services.

RESTful Web Services have three main characteristics

- The services use HTTP methods explicitly

- The services are accessible through Uniform Resource Identifiers (URIs) and

- The services are stateless

The Oracle Database Cloud allows access to data within a particular Database Cloud Service through RESTful Web Services. Each Database Cloud Service includes a RESTful Web Service wizard which makes it easy to create RESTful Web Services for use by applications which reside outside the Database Cloud. In addition, the next release of SQL Developer will include the ability to create RESTful Web Services from SQL Developer worksheets.

Please be aware that the RESTful capabilities described in this white paper are available through the APEX Listener, so most of the information contained within also applies to Oracle Databases access through the APEX Listener outside the Oracle Database Cloud.

Architecture of Oracle Database Cloud RESTful Web Services

You define RESTful Web Services for your Database Cloud Service by using a wizard in your Service, which is described later in this paper.

There are three levels of organization used for RESTful Web Services, whose characteristics and attributes are described in the following sections.
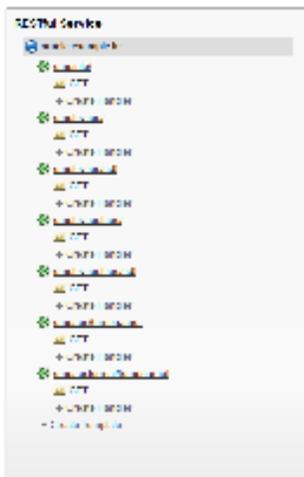
The RESTful Service Module

The RESTful Service module is used to group Services and to implement RESTful security. A module is a container used to group Services which will commonly be used together or which have specific security grants associated with all the RESTful calls within the module. All export and import of Services is done at the module level.

The attributes of a module are

- Name – a unique identifier for the module

- URI Prefix – used, along with the URI Template (described below) and the URI of the Database Cloud Service, to uniquely identify an individual RESTful Web Service.

- Origins allowed – prevents potential cross-site scripting issues for non-public RESTful Web Services. You can list the origins, other than the home domain, which can call the handlers in this module.

- Required privilege – designates whether a module is public, with no privilege specified, or requires the caller to have a particular privilege. Privileges are described in the section below on security.

- Pagination size – the default number of rows returned per page for all handlers with a Query or Feed source type. You can override this default with an attribute in the template definition, described below.

- Status – allows you to designate whether a module is available (Published) or not (Not Published) to callers

Templates

A template is identified by a URI, which is comprised of portions also based on the Database Cloud Service and the URI prefix of the module. If you want to pass variables which will be used as bind variables in the SQL or PL/SQL source for the Service, you add them after the final / in the URI template field between two curly braces. The name specified between the braces is used as the bind variable in the Source code. For instance, if you defined the URI template as

    emp/{empid}

you could use empid as a bind variable in your source code. You can also use standard notation such as

    emp/?dept={deptno}

in the template URL to use deptno in your source code. You might choose to do this in order to leave the base URI for the RESTful Web Service as ending with emp/, for use as a Feed source type, described below.

You can indicate a entity tag for a template, which is used as an identifier in the HTTP Header and uniquely identifies a particular version of the data returned from a RESTful Web Service call. With this tag, you can identify results returned by different calls of the same RESTful Web Service. The default for this attribute is Secure HASH, although you can also specify your own query or use the entry None to indicate that no entity tag will be generated for the resource template

You can also set the priority for a template. This priority is used if you have two templates with the same URI signature. If there are multiple templates meeting this condition, the APEX Listener orders them based on the specified priority, with the highest number being the first evaluated. If the appropriate handler is found in a template, no further evaluations are performed. If not, the Listener moves on to the next handler or, if no handlers remain, returns an error.

Handlers

A handler is based on a specific HTTP method, such as GET, POST, PUT or DELETE. You can only have one handler for each HTTP method for each template.

The attributes of a handler are
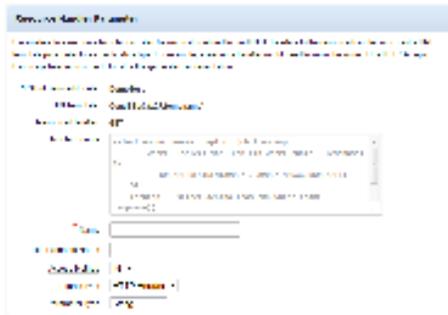
- Method – indicates which HTTP method is used for this handler

- Source Type – indicates which source type is used for this handler, if the handler uses a GET method. All other methods only allow for PL/SQL source types. Source types are described in more detail below.

- Requires Secure Access – specifies whether the URI is allowed using the HTTP or HTTPS protocol.

- Pagination – overrides the default pagination set in at the module level.

- Source – contains either an SQL statement or a PL/SQL anonymous block, depending on the Source Type.

If a handler uses a GET method, a Test button will appear below the Source window. This button simply calls the handler's URI. You can set value for bind variables in the URI with a button next to the Test button. For other methods, you will have to use an external client of some type. You can download a RESTful client that can be used as a page in an APEX application, an instance of the APEX Listener or various application servers. You can find the code and documentation for this client on Oracle Technology Network [here](#).

You can also define parameters, using the section at the bottom of the handler definition page, as shown here. Parameters are defined to specify a data type for incoming parameters, bind an HTTP header to a bind variable or to define outbound parameters.

There are two special OUT parameters which are used when you perform data manipulation (INSERT, UPDATE or DELETE) operations in a RESTful Web Service, which is described in a section below.
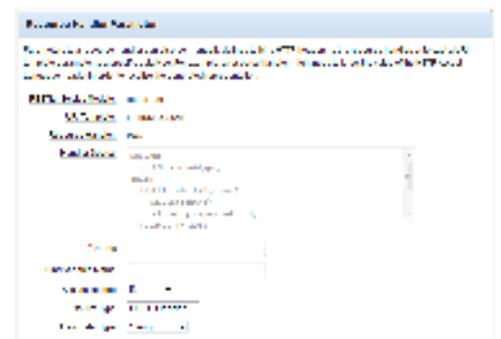
Source Types of RESTful Web Services

When you create a RESTful Web Service based on a GET method, you have the option to choose different Source Types. The Source Type indicates the way the return values from the RESTful Web Service are formatted automatically.

There are two basic categories of Source Types – SQL and PL/SQL, and a total of 5 distinct Source Types.

SQL Source Types

The Query, Query One Row, Feed and Media Resource Source Types use SQL to define the data returned for a RESTful Web Service call.

Query – A Query Source Type is defined as any standard SQL query[1]. The values returned from a Query RESTful Web Service are marshaled as a set of column values in JSON format or as a comma-separated variable (.csv) file.

Query One Row – The Query One Row Source Type only returns a single row from a SELECT operation. This Source Type is designed to test for the existence of a row which meets the selection criteria specified in the SQL statement. If a query with this Source Type does not find a result, a 404 Not Found error is returned.

Feed – A Feed Source Type returns both data and a URI for each row that is generated from an SQL query, based on a particular structure for the SQL query. The URL returned by the Feed Web Service is formed by using the URI for the Web Service with the value for the first column in the SQL statement passed as the argument in the URI. This URI is used to call another RESTful Web Service which accepts this argument, typically for more detailed information. As an example, a RESTful Web Service with a Feed Source Type that used the following URI

http://database.oracle.com/apex/hr/employees/

would include a URI with this form, based on a value for the initial column of '1234'

http://database.oracle.com/apex/hr/employees/1234

Media Resource – With each of the three types described above, the source for the data was a SQL statement, and the results from that SQL statement would be formatted by the Oracle Database Cloud Service to be

---

[1] You can use the PL/SQL source type, described below, to implement other types of SQL statements, such as INSERT, UPDATE and DELETE.

returned as either JSON or a .csv file. The Media Resource Source Type does not allow that formatting option to be implemented. This Source type takes a single SQL statement with the format of

SELECT '*content_type*', *column* FROM . . .

where '*content_type*' is a string passed to the browser to be used to identify the incoming data, and the *column* to identify the source of the data being sent back. This data is sent back untouched by the Oracle Database Cloud Service. The Media Resource Source Type is typically used for media objects, such as images, where the data will be directly handled by the recipient making the call.

PL/SQL Source Type

The PL/SQL Source Type allows you to use PL/SQL code to create and return data from a RESTful Web Service call. The PL/SQL within this Source Type is essentially an anonymous PL/SQL block. You can use a PL/SQL Source Type to implement SQL DML, extended logic or to format and return data in any way that you need. The APEX Listener does not modify returns from a handler with a PL/SQL Source Type.

Calling RESTful Web Services

You call a RESTful Web Service using the unique URI for the Service. This URI is composed of the URI for the database server followed by /*apex*, the URI prefix from the module definition and the URI portion from the template. For instance, if the server URI was *database-demo.yourcompany.com*, the URI prefix was *sample* and the template URI was *emp*, the call would be

http://database-demo.yourcompany.com/apex/sample/emp

For RESTful queries which use the GET method, you can click on the Test button, which is just below the text window where you enter you SQL or PL/SQL. The RESTful Web Service will be called in your browser, and you can simply copy the URI from the address window.

You can also use this method to get the base URI for data manipulation, described in the next section, but you will typically have to use JSON in the body of the HTTP request with these types of calls, as you will see.

Data manipulation with RESTful Web Services

You can also use RESTful Web Services to write, modify and delete data in your Database Cloud Service. These operations are performed with the use of the PUT, POST and DELETE methods.

All of these operations use PL/SQL code in the RESTful Web Service definition. The following sections contain sample code to help you understand the concepts of performing these operations. However, for more complete code examples please see these [samples](), available from the Oracle Technology Network.

DELETE operations

A DELETE operation would normally be called with a DELETE method for the handler. In the following example, the URI for the RESTful Web Service would end with a variable that would be passed to the Service to indicate which row should be deleted, as with

```
...emp/{empid}
```

The PL/SQL code for this RESTful Web Service could be as simple at this

```
BEGIN

    DELETE FROM emp WHERE empno = :empid;

END;
```

Normally, you would also send back status information with an outbound parameter, which will be discussed in the section on INSERTs.

Please note that the DELETE method is *idempotent*. Idempotent means that the method can be run many times resulting in the same final state on the server. If a DELETE RESTful Web Service successfully deletes a row, the row will no longer be in the table. If the RESTful Web Service does not delete the row, the reason is that the row identified with the passed unique identifier is not in the table – which is the same final state as if the DELETE were successful.

UPDATE operations

Update operations are typically mapped to a PUT method. As with the previous DELETE Service, you would pass a unique identifier for the row you wish to update as part of the RESTful Web Service URI. The PL/SQL code for the RESTful Web Service would look something like this –

```
BEGIN

    UPDATE emp SET sal = :sal WHERE empno = :empid;

END;
```

In this example, the bind variable sal is not passed as part of the URI. This bind variable is included as JSON in the body of the request. The APEX Listener automatically converts all values passed in this manner into bind variables with the name specified as part of the JSON name-value pair. If you wanted to pass two variables in this way, you would use JSON formatting like this –

```
{"sal":"10000","name":"Tucker"}
```

The PUT method is also idempotent, but, unlike the DELETE method mapped to the DELETE operation, this characteristic is not always true. For instance, if you were going to set the value of SAL to a specific value, you could call this method multiple times and the value would still be that specific value. However, if your UPDATE statement were something like this –

```
UPDATE emp SET sal = (sal * 1.1) WHERE empno = :empid;
```

the value would obviously be different with each invocation of the method.

INSERT operations

The POST method is normally mapped to the INSERT operation, since this operation is not normally idempotent – if you call an INSERT 5 times, you want to perform the INSERT operation 5 times.

In this example, you will typically not pass a unique identifier to the RESTful Web Service, as the best practice is to have that identifier assigned on the database by a trigger.

The PL/SQL code for implementing this Service is slightly more complex, as it includes the use of a specific outbound variables.

```
DECLARE

  id EMP.empno%type;

BEGIN

  INSERT into EMP(ename)

   values(:ENAME)

   returning empno into id;

  :status := 201;

  :location := id;

  commit;

END;
```

In this code sample, you return the value of 201, which is the HTTP status code for CREATED, in the status variable. In addition, you pass back the id assigned to the new row in the location variable.

You will have to define these variables as part of the handler definition, as described above. The bind variable names would be status and location, respectively. The name for the status bind variable will be X-APEX-STATUS-CODE, a specific outbound variable understood by the APEX Listener.

The location bind variable will be assigned the name X-APEX-FORWARD, another specific outbound variable, but one that provides more functionality. The APEX Listener expects this variable to contain a unique identifier for a resource, and it uses this variable as a Feed Source Type would to return the row that was just inserted into the database.

For instance, if an invocation of this RESTful Web Service call

```
. . .empInsert/
```

returned the value 8001 in the X-APEX-FORWARD variable, the APEX Listener would call this RESTful Web Service for the values to return

```
. . .empInsert/8001
```

Security and RESTful Web Services

An Oracle Database Cloud Service is based on a single schema, and all RESTful Web Services which access data in this schema are run as the owner of the schema. Without security limitations, all RESTful Web Services are public, which means that anyone who has the URL of the RESTful Web Service call can use the call.

You can add security at the module level for RESTful Web Services. This security is separate from security in an Oracle Database, as this access level controls who can execute the RESTful Web Service. You do not place security on the data itself, since all RESTful Web Services are executed as the schema owner with all privileges[2].

The way to implement security on your RESTful Web Services is to define a RESTful Service Privilege and assign it to one or more RESTful Web Service modules.

There is another method of implementing security which could be used. The APEX Listener, which implements access to RESTful Web Services, runs in a web server. If the web server running the APEX Listener allows you to define access limitations, you could simply prevent users from accessing the web server itself, which would also block any RESTful Web Services which used the APEX Listener in the web server.

You can also choose to add logic in the actual handlers to deliver data based on user identity or some other variable included with the RESTful Web Service request, in the same way you would use any other variable limit access.
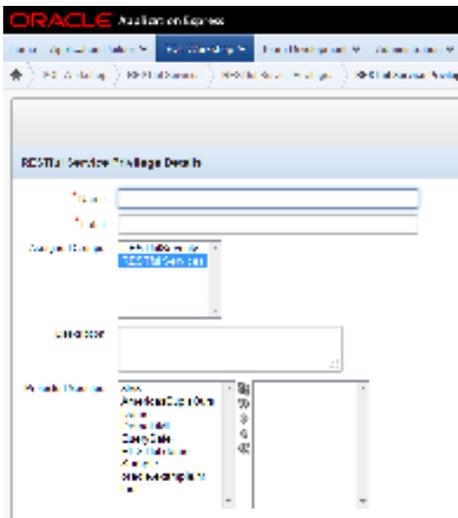
Using RESTful Web Service Privileges

RESTful Web Service Privileges are defined through a wizard which is available through a link in the Tasks box on the right hand side of the home page for RESTful Web Services within the SQL Workshop area of your Application Express environment. You can define a new RESTful Service Privilege by clicking on the Create button on the home page for these privileges, which will bring up a wizard pages as partially shown here.

Each RESTful Service Privilege has a unique name and label. The Privilege is based on a user group defined within the APEX environment – all members of the user groups assigned to the Privilege have access to

---

[2] Outside of the Oracle Database Cloud, you have more flexibility, as you could implement a RESTful Web Service which accessed data in another schema. You would be executing the RESTful Web Service as the owner of the schema in which the Service was defined, and that user would be subject to the standard access privileges used in an Oracle Database.

all Web Services defined in modules limited by that Security Privilege.

For convenience, you can assign a privilege to one or more modules from this page, or you can add in the Privilege to individual modules.

The Security Privilege allows users within those user groups to execute the RESTful Web Services within a module. You have two different ways to authenticate RESTful Web Service users – first party authentication and third party authentication. Both of these methods are described in more detail in the RESTful Services Developers Guide, available on Oracle Technology Network in the APEX Listener area of the Developer Tools section, currently located [here](#).

First party authentication

With first party authentication, you use the same environment that holds the RESTful Web Services to authenticate users. For the Oracle Database Cloud and other Oracle environments using the APEX Listener, that environment is APEX itself.

For first party authentication, you need to pass the application ID and the session ID for an APEX application session. You can include this information in the HTTP header of the RESTful Web Service request or as part of the URI for the request.

Third party authentication

If you are using RESTful Web Services from another, third party application, such as Java or a dynamic language like PHP, you will have to use third party authentication through OAUTH2, the emerging standard for third party authentication on the Internet. You have probably already experienced an OAUTH2 authentication used by applications accessing Facebook or other social media sites.

OAUTH2 authentication is a two step process. The first step is to register the third party application and receive a request token. This token allows the application to request authentication.

The second step is to request and authenticate a user of the application. In this step, the user is first prompted for their credentials and then asked if the third party application is allows to access the RESTful Web Service on their behalf. If these steps complete successfully, the application receives an access token that gives it the ability to use the Web Service on behalf of the user.

ORACLE

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

Oracle Cloud Computing
February 2014
Author: Rick Greenwald

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