Table of Contents

Developing Oracle RESTful Services

This tutorial will guide you through the process of creating a simple RESTful Service to GET some data from a database. Then we will protect the created resource using ORDS roles. More detailed description of developing and protecting Oracle RESTful Services can be found here:

https://oracle-base.com/articles/misc/oracle-rest-data-services-ords-create-basic-rest-web-services-using-plsql

https://oracle-base.com/articles/misc/oracle-rest-data-services-ords-autorest

https://oracle-base.com/articles/misc/oracle-rest-data-services-ords-authentication

Defining RESTful Services using AutoREST

First let's create a simple table and insert some data for our test:

```
CREATE TABLE USERS (
  USERID NUMBER(4,0),
  NAME VARCHAR2(10 BYTE),
  SURNAME VARCHAR2(10 BYTE),
  CONSTRAINT PK_EMP PRIMARY KEY (USERID)
);

insert into USERS (USERID,NAME,SURNAME) values (1,'Smith','Liam');
insert into USERS (USERID,NAME,SURNAME) values (2,'Clerk','John');
insert into USERS (USERID,NAME,SURNAME) values (3,'Boguski','Marek');
insert into USERS (USERID,NAME,SURNAME) values (4,'Vardy','Michael');
commit;
```

First step is to enable REST web services for the schema. We can specify any url mapping pattern for the schema

```
ORDS.ENABLE_SCHEMA(p_enabled => TRUE,
                   p_schema => 'MYSCHEMA',
                   p_url_mapping_type => 'BASE_PATH',
                   p_url_mapping_pattern => 'myschema',
                   p_auto_rest_auth => FALSE);
```

In the example above we map our schema to a certain url. Assuming that we run our test on the devdb11 database, the URL will be:

https://oraweb.cern.ch/ords/devdb11/myschema/

https://oraweb.cern.ch/ords/ - this part of the URL comes from the internal configuration of ORDS that is managed by IT-DB.

/devdb11 - depends on the database used.

The next step is to define a service for one of our tables (using AutoREST) or map a specific URL to our statement (select, procedure...). In this example we use the AutoREST feature to expose the whole table for READ, UPDATE, INSERT and DELETE.

```
ORDS.ENABLE_OBJECT(p_enabled => TRUE,
                   p_schema => 'myschema',
                   p_object => 'USERS',
                   p_object_type => 'TABLE',
                   p_object_alias => 'users',
```

```
    p_auto_rest_auth => FALSE);
```

p_object_alias maps to the last part of the URL under which we can access our resource:

https://oraweb.cern.ch/ords/devdb11/myschema/users

Let's test our REST endpoint:

```
curl --basic https://oraweb.cern.ch/ords/devdb11/myschema/users/

{"items":[{"userid":1,"name":"Smith","surname":"Liam","links":[{"rel":"self","href":"https
```

UPDATE, INSERT and DELETE operations using AutoREST feature are described here:

https://oracle-base.com/articles/misc/oracle-rest-data-services-ords-autorest

Defining RESTful Services using PL/SQL

In this example we will define our own PL/SQL statement to get the date from the USERS table. We already have our schema rest-enabled (look at ORDS.ENABLE_SCHEMA above).

We define a simple service:

```
BEGIN
  ORDS.define_service(
    p_module_name    => 'testmodule',
    p_base_path      => 'users/',
    p_pattern        => 'name/',
    p_method         => 'GET',
    p_source_type    => ORDS.source_type_query,
    p_source         => 'SELECT USERID, NAME FROM USERS',
    p_items_per_page => 0);

  COMMIT;
END;
```

p_base_path and p_pattern will become parts of our URL:

https://oraweb.cern.ch/ords/devdb11/myschema/users/name/

Let's try our service:

```
curl --basic https://oraweb.cern.ch/ords/devdb11/myschema/users/name/

{"items":[{"userid":1,"name":"Smith"},{"userid":2,"name":"Clerk"},{"userid":3,"name":"Bogu
```

UPDATE, INSERT and DELETE operations are described here:

https://oracle-base.com/articles/misc/oracle-rest-data-services-ords-create-basic-rest-web-services-using-plsql

Protecting RESTful Services

Basic Authentication

To protect our service we have to define a role, a privilege and a mapping between the privilege and the URL we want to protect.

ORDS roles map to e-groups but e-group name for protecting ORDS RESTful Services must follow the pattern `ords-rest-access-*`

In this example we created and are members of ords-rest-access-ims e-group at CERN.

First we create a role with the same name:

```
BEGIN
  ORDS.create_role(
    p_role_name => 'ords-rest-access-ims'
  );

  COMMIT;
END;
```

p_role_name directly maps to the CERN e-group name.

Now we can create a privilege associated with that role:

```
DECLARE
  l_arr OWA.vc_arr;
BEGIN
  l_arr(1) := 'ords-rest-access-ims';

  ORDS.define_privilege (
    p_privilege_name => 'rest-access-test',
    p_roles          => l_arr,
    p_label          => 'Users data',
    p_description    => 'Allow access to Users data.'
  );

  COMMIT;
END;
```

p_roles param has to contain the role we defined above (can contain more than one role). We also specified the privilege name (p_privilege_name) and a description.

To protect our service we associate the privilege with our url mapping:

```
BEGIN
  ORDS.create_privilege_mapping(
    p_privilege_name => 'rest-access-test',
    p_pattern => '/users/*'
  );

  COMMIT;
END;
```

after creating the mapping we can try to access our service:

```
curl --basic https://oraweb.cern.ch/ords/devdb11/myschema/users/
```

We should receive error 401 Unauthorized.

Let's try passing our credentials. In this example `dmoskali` user belongs to e-group ords-rest-access-ims

```
curl --basic --user dmoskali https://oraweb.cern.ch/ords/devdb11/myschema/users/

Enter host password for user 'dmoskali':
{"items":[{"userid":1,"name":"Smith","surname":"Liam","links":[{"rel":"self","href":"https
```

To protect your services you can create any e-group that starts with the prefix **ords-rest-access-**

ORDS & OAuth 2 : Client Credentials flow

The client credentials flow is a two-legged process. For this flow we use the client credentials to return an **access token**, which is used to authenticate calls to protected resources.

First step is to **register** your application as a OAUTH **client**:

```
BEGIN
  OAUTH.create_client(
    p_name            => 'pcitdes12',
    p_grant_type      => 'client_credentials',
    p_owner           => 'lurodrig',
    p_description     => 'Luis client for demo OAUTH2 on ORDS. It will CRUD operations on
    p_support_email   => 'lurodrig@cern.ch',
    p_privilege_names => 'crud-operations-on-site-table'
  );
  COMMIT;
END;
```

- p_name: name of your application/client. It must be unique. Here I am using the name of my terminal.
- p_grant_type: for this flow we want to use **client_credentials** grant. The client will need to present his credentials, **client_id** & **client_secret** to access the **access token**.
- p_owner: here I will set my NICE username for clarity.
- p_description: something meaningful, if possible...
- p_support_email: self-explained, I hope...
- p_privilege_names: this field will tell ORDS what actions this client wants to do.

You can find the full specification for the OAUTH PL/SQL package here

Like in the basic authentication scenario we have to **create a role**:

```
BEGIN
  ORDS.create_role(
    p_role_name => 'cerndb-mwctl-a-01-dev-site-api'
  );

  COMMIT;
END;
```

We have to **link** that **role** with the **privileges**:

```
DECLARE
  l_priv_roles owa.vc_arr;
BEGIN
  l_priv_roles(1) := ''ords-rest-access-dev-site-api';

  ords.define_privilege(
    p_privilege_name     => 'crud-operations-on-site-table',
    p_roles              => l_priv_roles,
    p_label              => 'Sites CRUD',
    p_description        => 'Provides the ability to create, ' ||
                            'update and delete sites '
  );
END;
COMMIT;
```

The last step will be **grant** our **client** with that **role**:

```
BEGIN
  OAUTH.grant_client_role(
    p_client_name => 'pcitdes12',
    p_role_name   => 'ords-rest-access-dev-site-api'
  );

  COMMIT;
END;
```

And we are done! Now we have to retrieve the **client_id** and **client_secret**:

```
SELECT id, name, client_id, client_secret
FROM   user_ords_clients;
```

With the **CLIENT_ID** and **CLIENT_SECRET** you can request the access_token to ORDS:

```
curl --basic --user CLIENT_ID:CLIENT_SECRET --data "grant_type=client_credentials" https:/
```

The output will look like this:

```
{"access_token":"cXGjuDYwwmn49nVAOoistA..","token_type":"bearer","expires_in":3600}
```

Finally with the **access_token** you can access your protected resources:

```
curl -H "Authorization: Bearer cXGjuDYwwmn49nVAOoistA.." https://oraweb.cern.ch/ords/devdb
```

ORDS & OAuth 2 : Implicit

The OAuth2 Implicit grant flow is suitable for client-side web applications (for instance a JavaScript-client), since we don't need to store the client_id and client_secret.

It requires user interaction instead of storing the credentials. The user accesses an URL in a browser and the browser prompts for credentials.

After the authentication, the browser is redirected to a page which include an access token as a parameter in the URL. Than we can use the **access_token** to access protected resources.

Prerequisite: We start by creating an ORDS-role (p_role_name) as explained in the Basic authentication -part of this twiki. Later you will need to associate your client with this ORDS-role.

Secondly we must **remember to clean up the OAUTH metadata**. This is done in the following way:

```
BEGIN
  OAUTH.revoke_client_role(
    p_client_name => 'pcitdb35',
    p_role_name   => 'ords-rest-access-cerndb-rota-users-api'
  );
  COMMIT;
END;
/

BEGIN
  OAUTH.delete_client(
    p_name => 'pcitdb35'
  );
  COMMIT;
END;
/
```

Now we define the OAuth2 Implicit grant flow. The first step is to **create the client** with the grant type implicit :

```
BEGIN
  OAUTH.create_client(
    p_name            => 'pcitdb35',
    p_grant_type      => 'implicit',
    p_owner           => 'ralvsvaa',
    p_description     => 'A client for DB Rota tool',
    p_redirect_uri    => 'https://oraweb.cern.ch/ords/devdb11/db-rota/users/redirect',
    p_support_email   => 'ralvsvaa@cern.ch',
    p_support_uri     => 'https://oraweb.cern.ch/ords/devdb11/db-rota/users/support',
    p_privilege_names => 'ords-rest-access-crud-operations-on-users-table'
  );

  COMMIT;
END;
/
```

Secondly display the client details and **retrieve the client_id**:

```
COLUMN name FORMAT A20

SELECT id, name, client_id, client_secret
FROM   user_ords_clients;
```

We will get a table showing the client details. The **client_id** will be used in the later steps.

Associate the created **client** with the **ORDS-role** from the Basic Authentication -part. This role holds the correct privileges for the resources it needs to access:

```
BEGIN
  OAUTH.grant_client_role(
    p_client_name => 'pcitdb35',
    p_role_name   => 'ords-rest-access-cerndb-rota-users-api'
  );
  COMMIT;
END;
/
```

Display the client-role relationship:

```
COLUMN client_name FORMAT A30
COLUMN role_name FORMAT A20

SELECT client_name, role_name
FROM   user_ords_client_roles;
```

Now we need to request an **access_token**. This is done by accessing the following URL from a browser:

https://oraweb.cern.ch/ords/devdb11/db-rota/oauth/auth?response_type=token&client_id={client_id}&state={random-

It's important to set the response_type to **token**, and insert your specific schema (p_url_mapping_pattern, e.g. 'db-rota'), the **client_id** and a random string in the URL.

We should get an Error 401 Unauthorized. From the 'Sign in'-link on this page, we can sign in with the credentials connected to the **e-group** (p_role_name).

The e-group was defined in the 'Basic Authentication'.

Next we will be directed to an approval page. Click the "Approve" button, which will take us to the redirect page we specified for the client.

We should get an Error 500 since the redirect URL is not an existing page. In the URL we will find the **access_token** as one of the parameters.

And we are done! Now we have the **access_token** and can access the protected resource:

```
curl -H "Authorization: Bearer {access_token}" https://oraweb.cern.ch/ords/devdb11/db-rota
```

-- RebekkaAlvsvaag - 2017-07-06

How to see defined enpoints, privileges, roles...

Once you REST-enabled your schema you can get the metacatalog that describes all modules/restpoints defined:

https://devapex-sso.cern.ch/ords/devdb11/myschema/metadata-catalog/

Or you can query directly user_ords_modules, user_ords_templates, user_ords_handlers tables.

To see defined roles:

```
SELECT id, name
FROM   user_ords_roles;
```

Display privileges defined:

```
SELECT id, name
FROM   user_ords_privileges;
```

Display privilege-role relation:

```
SELECT privilege_id, privilege_name, role_id, role_name
FROM   user_ords_privilege_roles;
```

Display privilege mappings:

```
SELECT privilege_id, name, pattern
FROM   user_ords_privilege_mappings;
```

How to get the HTTP status of the operation from PL/SQL

To decide the HTTP status of the operation from PL/SQL, you have to do a couple of things.

- Declare an (INTEGER) OUT parameter in your procedure, so that you can pass back the desired status. For instance

```
PROCEDURE deleteExternalVisitors(p_data IN BLOB,p_status OUT INTEGER)
```

. Write your own pl/sql code and decide what HTTP status you want to pass back in p_status (a list of HTTP status codes can be found at: http://www.restapitutorial.com/httpstatuscodes.html)

- Then define the handler calling this procedure, for instance

```
BEGIN A3_API.deleteExternalVisitors(p_data => :body, p_status => :status); END;
```

- Finally define a parameter for your handler, as in

https://docs.oracle.com/cd/E56351_01/doc.30/e56293/ORDS-reference.htm#AELIG90210

In order to set the HTTP status, use X-APEX-STATUS-CODE as indicated in https://docs.oracle.com/cd/E21611_01/doc.11/e21058/rest_api.htm#AELIG7137

- Example

```
BEGIN
  ORDS.DEFINE_PARAMETER(
    p_module_name => 'module_ws',
    p_pattern => 'bookings/',
    p_method => 'DELETE',
    p_name => 'X-APEX-STATUS-CODE',
    p_bind_variable_name => 'status',
    p_source_type => 'HEADER',
    p_access_method => 'OUT'
  );
COMMIT;
END;
```

That s it. ORDS will send the HTTP status that you have determined in your code (received as p_status) as it maps to your parameter status (in the handler definition) that maps to X-APEX-STATUS-CODE.

You have to define a parameter for each combination of pattern and method.

How to deal with BLOBs

ORDS does not provide anymore the file upload and download parameters. Nevertheless if you need to download and upload binaries to the database you can always develop your own little API for this purpose.

Download/Upload API example

Imagine that you have a very simple table like this one

```
CREATE TABLE "IMAGE"
  ( "ID" NUMBER NOT NULL ENABLE,
    "BIN" BLOB
  );
ALTER TABLE "IMAGE" ADD CONSTRAINT "IMAGE_PK" PRIMARY KEY ("ID") ;
COMMIT;
```

The first things that you have to do is to enable ORDS in your schema, define a module and a template:

```
BEGIN
 ORDS.DEFINE_MODULE(
      p_module_name    => 'module1',
      p_base_path      => '/app/images/',
      p_items_per_page =>  25,
      p_status         => 'PUBLISHED',
      p_comments       => NULL);
COMMIT;
END;

BEGIN
  ORDS.DEFINE_TEMPLATE(
      p_module_name    => 'module1',
      p_pattern        => 'image/:id',
      p_priority       => 0,
      p_etag_type      => 'HASH',
      p_etag_query     => NULL,
      p_comments       => NULL);
COMMIT;
END;
```

Download

In order to simplify the code we are using the ORDS Media Source Type. With this type you just need to specify in your query the returning type (image/png) and the binary value:

```
BEGIN
  ORDS.DEFINE_HANDLER(
      p_module_name    => 'module1',
      p_pattern        => 'image/:id',
      p_method         => 'GET',
      p_source_type    => ORDS.source_type_media,
      p_items_per_page =>  0,
      p_mimes_allowed  => '',
      p_comments       => NULL,
      p_source         => 'SELECT ''image/png'', BIN FROM IMAGE where id = :id'
      );
COMMIT;
END;
```

Try it! `https://oraweb.cern.ch/ords/devdb11/app/lurodrig/images/image/1`

Upload

For uploading you can write a little PL/SQL block like the below one. The `:body` parameter refers to the HTTP request body

```
BEGIN
  ORDS.DEFINE_HANDLER(
      p_module_name    => 'module1',
      p_pattern        => 'image/:id',
      p_method         => 'POST',
      p_source_type    => ORDS.source_type_plsql,
      p_items_per_page =>  0,
      p_mimes_allowed  => '',
      p_comments       => NULL,
      p_source         =>
' DECLARE
   P_BIN BLOB := :body;
   P_ID INTEGER := :id;
  BEGIN
   UPDATE IMAGE SET BIN = P_BIN WHERE ID = P_ID;
  END;');
COMMIT;
```

```
END;
```

Try it!

```
curl -v -F "file=@image.jpg" https://devoraweb.cern.ch/ords/devdb11/lurodrig/app/images/ima
```

References

- This article from thatjeffsmith.com is very complete.
- And of course Tim Hall always has very good stuff, check out this article

-- DamianRadoslawMoskalik - 2016-09-22

This topic: DB > DevelopingOracleRestfulServices
Topic revision: r21 - 2019-10-07 - NiloChinchilla