Oracle® REST Data Services
Installation, Configuration, and Development Guide
Release 3.0
E56293-05

October 2015
Explains how to install and configure Oracle REST Data Services, and to develop applications that use Oracle REST Data Services.
2 Configuring Oracle REST Data Services (Advanced)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Configuring Multiple Databases</td>
<td>2-1</td>
</tr>
<tr>
<td>2.1.1 About the Request URL</td>
<td>2-1</td>
</tr>
<tr>
<td>2.1.2 Configuring Additional Databases</td>
<td>2-2</td>
</tr>
<tr>
<td>2.1.3 Routing Based on the Request Path Prefix</td>
<td>2-2</td>
</tr>
<tr>
<td>2.1.3.1 Example of Routing Based on the Request Path Prefix</td>
<td>2-3</td>
</tr>
<tr>
<td>2.1.4 Routing Based on the Request URL Prefix</td>
<td>2-3</td>
</tr>
<tr>
<td>2.1.4.1 Example of Routing Based on the Request URL Prefix</td>
<td>2-3</td>
</tr>
<tr>
<td>2.2 Configuring Security, Caching, Pre- and Post Processing, Environment, and Excel Settings</td>
<td>2-4</td>
</tr>
<tr>
<td>2.3 Developing RESTful Services for Use with Oracle REST Data Services</td>
<td>2-4</td>
</tr>
</tbody>
</table>

3 Developing Oracle REST Data Services Applications

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Introduction to Relevant Software</td>
<td>3-2</td>
</tr>
<tr>
<td>3.1.1 About Oracle Application Express</td>
<td>3-2</td>
</tr>
<tr>
<td>3.1.2 About RESTful Web Services</td>
<td>3-2</td>
</tr>
<tr>
<td>3.2 Getting Started with RESTful Services</td>
<td>3-2</td>
</tr>
<tr>
<td>3.2.1 RESTful Services Terminology</td>
<td>3-2</td>
</tr>
<tr>
<td>3.2.2 Accessing RESTful Services Using Application Express</td>
<td>3-3</td>
</tr>
<tr>
<td>3.2.3 &quot;Getting Started&quot; Documents Included in Installation</td>
<td>3-4</td>
</tr>
<tr>
<td>3.2.4 Exploring the Sample RESTful Services (Tutorial)</td>
<td>3-4</td>
</tr>
<tr>
<td>3.2.5 About cURL and Testing RESTful Services</td>
<td>3-9</td>
</tr>
<tr>
<td>3.3 Automatic Enabling of Schema Objects for REST Access (AutoREST)</td>
<td>3-9</td>
</tr>
<tr>
<td>3.3.1 Examples: Accessing Objects Using RESTful Services</td>
<td>3-10</td>
</tr>
<tr>
<td>3.3.1.1 Get Schema Metadata</td>
<td>3-11</td>
</tr>
<tr>
<td>3.3.1.2 Get Object Metadata</td>
<td>3-12</td>
</tr>
<tr>
<td>3.3.1.3 Get Object Data</td>
<td>3-13</td>
</tr>
<tr>
<td>3.3.1.4 Get Table Data Using Paging</td>
<td>3-14</td>
</tr>
<tr>
<td>3.3.1.5 Get Table Data Using Query</td>
<td>3-15</td>
</tr>
<tr>
<td>3.3.1.6 Get Table Row Using Primary Key</td>
<td>3-17</td>
</tr>
</tbody>
</table>
4.3 Adding and Removing a NoSQL Store for Use with Oracle REST Data Services .......... 4-6
4.4 NoSQL REST Services .................................................................................................. 4-6
4.4.1 Metadata Services .................................................................................................. 4-7
4.4.1.1 Read Metadata (GET) .................................................................................... 4-7
4.4.2 Data Services ......................................................................................................... 4-10
4.4.2.1 Read Data (GET) ............................................................................................ 4-10
4.4.2.2 Idempotent Write (PUT) ................................................................................. 4-15
4.4.2.3 Write (POST) .................................................................................................. 4-15
4.4.2.4 Delete (DELETE) .............................................................................................. 4-16
4.4.3 DDL Services ........................................................................................................ 4-16
4.4.3.1 Create/Update (POST) .................................................................................. 4-16
4.4.3.2 Drop (POST) .................................................................................................. 4-17
4.4.3.3 Poll (GET) ...................................................................................................... 4-17
4.4.4 Common Parameters ............................................................................................. 4-18
4.4.4.1 Parameters to Skip Results and Limit the Number of Results ......................... 4-18
4.4.4.2 Parameters to Specify Query Conditions ......................................................... 4-19
4.4.4.3 Parameters to Specify Direction (Order), and Consistency and Durability Guarantees 4-19

5 ORDS PL/SQL Package Reference

ORDS.CREATE_ROLE ........................................................................................................ 5-2
ORDS.CREATE_SERVICE ................................................................................................. 5-3
ORDS.DEFINE_HANDLER .................................................................................................. 5-4
ORDS.DEFINE_MODULE .................................................................................................... 5-6
ORDS.DEFINE_PARAMETER ............................................................................................... 5-7
ORDS.DEFINE_PRIVILEGE ............................................................................................... 5-8
ORDS.DEFINE_SERVICE .................................................................................................... 5-10
ORDS.DEFINE_TEMPLATE .................................................................................................. 5-13
ORDS.DELETE_MODULE .................................................................................................... 5-14
ORDS.DELETE_PRIVILEGE ............................................................................................... 5-15
ORDS.DELETE_ROLE ........................................................................................................ 5-16
ORDS.DROP_REST_FOR_SCHEMA ................................................................................... 5-17
ORDS.ENABLE_OBJECT .................................................................................................... 5-18
ORDS.ENABLE_SCHEMA .................................................................................................. 5-19
ORDS.PUBLISH_MODULE .................................................................................................. 5-20
ORDS.RENAME_MODULE ................................................................................................. 5-21
ORDS.RENAME_PRIVILEGE ............................................................................................. 5-22
ORDS.RENAME_ROLE ....................................................................................................... 5-23
ORDS.SET_MODULE_ORIGINS_ALLOWED ........................................................................ 5-24
ORDS.SET_URL_MAPPING ............................................................................................... 5-25

6 OAUTH PL/SQL Package Reference

OAUTH.CREATE_CLIENT ................................................................................................. 6-2
OAUTH.DELETE_CLIENT ................................................................................................. 6-3
A About the Oracle REST Data Services Configuration Files
   A.1 Locating Configuration Files ................................................................. A-1
   A.2 Setting the Location of the Configuration Files ................................. A-1
   A.3 Understanding the Configuration Folder Structure .......................... A-1
   A.4 Understanding the Configuration File Format ................................. A-2
      A.4.1 Understanding the url-mapping.xml File Format ....................... A-2
   A.5 Understanding Configurable Parameters ......................................... A-3

B Troubleshooting Oracle REST Data Services
   B.1 Enabling Debug Tracing ........................................................................ B-1
   B.2 Enabling Detailed Request Error Messages ........................................ B-1
   B.3 Configuring Application Express Static Resources with Oracle REST Data Services .... B-1

C Development Tutorial: Creating an Image Gallery
   C.1 Before You Begin ................................................................................ C-1
      C.1.1 About URIs .................................................................................. C-1
      C.1.2 About Browser Support ............................................................... C-2
      C.1.3 Creating an Application Express Workspace ............................... C-2
   C.2 Creating the Gallery Database Table ................................................ C-2
   C.3 Creating the Gallery RESTful Service Module .................................. C-3
   C.4 Trying Out the Gallery RESTful Service ........................................... C-4
   C.5 Creating the Gallery Application ....................................................... C-5
   C.6 Trying Out the Gallery Application ................................................... C-8
   C.7 Securing the Gallery RESTful Services ............................................. C-8
      C.7.1 Protecting the RESTful Services ................................................. C-8
      C.7.2 Modifying the Application to Use First Party Authentication ......... C-8
   C.8 Accessing the RESTful Services from a Third Party Application ....... C-9
      C.8.1 Creating the Third Party Developer User .................................... C-10
      C.8.2 Registering the Third Party Application ...................................... C-11
      C.8.3 Acquiring an Access Token ......................................................... C-12
      C.8.4 Using an Access Token ............................................................... C-12
      C.8.5 About Browser Origins .............................................................. C-13
      C.8.6 Configuring a RESTful Service for Cross Origin Resource Sharing .... C-14
      C.8.7 Acquiring a Token using the Authorization Code Protocol Flow .... C-15
         C.8.7.1 Registering the Client Application ....................................... C-15
         C.8.7.2 Acquiring an Authorization Code ......................................... C-15
         C.8.7.3 Exchanging an Authorization Code for an Access Token ........ C-16
         C.8.7.4 Extending OAuth 2.0 Session Duration ................................. C-17
      C.8.8 About Securing the Access Token .............................................. C-18
D Using the Multitenant Architecture with Oracle REST Data Services

D.1 Understanding the Installation Choices ................................................................. D-1
D.2 Installing Oracle REST Data Services into a CDB ............................................. D-2
D.3 Making All PDBs Addressable by Oracle REST Data Services (Pluggable Mapping) ... D-3
D.4 Uninstalling Oracle REST Data Services from a CDB ....................................... D-5

E Getting Started with RESTful Services

E.1 REST-Enable a Database Table ................................................................................ E-2
E.2 Create a RESTful Service from a SQL Query ...................................................... E-6
E.3 Protect Resources .................................................................................................... E-13
E.4 Register an OAuth Client Application ................................................................. E-14

Index
Oracle REST Data Services Installation, Configuration, and Development Guide explains how to install and configure Oracle REST Data Services. (Oracle REST Data Services was called Oracle Application Express Listener before Release 2.0.6.)

**Book Title and Scope Change:** Effective with Release 3.0, the title of this book is Oracle REST Data Services Installation, Configuration, and Development Guide. The addition of "Development" to the title reflects the fact that material from a previous separate unofficial "Developer's Guide" has been included in this book in Chapter 3, "Developing Oracle REST Data Services Applications".

**Topics:**
- Audience
- Documentation Accessibility
- Related Documents
- Conventions
- Third-Party License Information

**Audience**
This document is intended for system administrators or application developers who are installing and configuring Oracle REST Data Services. This guide assumes you are familiar with web technologies, especially REST (Representational State Transfer), and have a general understanding of Windows and UNIX platforms.

**Documentation Accessibility**

**Access to Oracle Support**
Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit [http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info](http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info) or visit [http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs](http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs) if you are hearing impaired.
Related Documents

For more information and resources relating to Oracle REST Data Services, see the following the Oracle Technology Network (OTN) site:


Conventions

The following text conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface</strong></td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td><em>italic</em></td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td><code>monospace</code></td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that is displayed on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>

Third-Party License Information

Oracle REST Data Services contains third-party code. See the Oracle Database Licensing Information book for notices Oracle is required to provide.

Note, however, that the Oracle program license that accompanied this product determines your right to use the Oracle program, including the third-party software, and the terms contained in the following notices do not change those rights.
Installing Oracle REST Data Services

This section describes how to install and deploy Oracle REST Data Services. (REST stands for Representational State Transfer.)

Name Change: Oracle REST Data Services was called Oracle Application Express Listener before Release 2.0.6.

Topics:
- About Oracle REST Data Services
- Understanding the Installation Process
- Installing and Configuring Oracle REST Data Services
- Running in Standalone Mode
- Deploying to Oracle WebLogic Server
- Deploying to GlassFish Server
- Deploying to Apache Tomcat
- Upgrading Oracle REST Data Services

1.1 About Oracle REST Data Services

Oracle REST Data Services is a Java EE-based alternative for Oracle HTTP Server and mod_plsql. The Java EE implementation offers increased functionality including a command line based configuration, enhanced security, file caching, and RESTful web services. Oracle REST Data Services also provides increased flexibility by supporting deployments using Oracle WebLogic Server, GlassFish Server, Apache Tomcat, and a standalone mode.

The Oracle Application Express architecture requires some form of web server to proxy requests between a web browser and the Oracle Application Express engine. Oracle REST Data Services satisfies this need but its use goes beyond that of Oracle Application Express configurations. Oracle REST Data Services simplifies the deployment process because there is no Oracle home required, as connectivity is provided using an embedded JDBC driver.

1.2 Understanding the Installation Process

This section offers an overview of Oracle REST Data Services and provides information about supported Java Platform, Enterprise Edition (Java EE) application servers and system requirements.
1.2.1 Supported Java EE Application Servers

Oracle REST Data Services supports the following Java EE application servers:

<table>
<thead>
<tr>
<th>Application Server</th>
<th>Supported Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle WebLogic Server</td>
<td>11g Release 1 (10.3.6) or later</td>
</tr>
<tr>
<td>GlassFish Server</td>
<td>Release 3.1.2 or later</td>
</tr>
<tr>
<td></td>
<td>For 4.0 (but not 4.1), the global switch for cdi 1-1 must be used to disable the injection check, as follows:</td>
</tr>
<tr>
<td></td>
<td>asadmin set configs.config.server-config.cdi-service.enable-implicit-cdi=false</td>
</tr>
<tr>
<td></td>
<td>However, it is strongly recommend that you <em>not</em> use 4.0, because setting that global switch affects all subsequent deployments on this web server, which may result in other deployments failing.</td>
</tr>
<tr>
<td>Apache Tomcat</td>
<td>Release 7.0.56 or later</td>
</tr>
</tbody>
</table>

1.2.2 System Requirements

Oracle REST Data Services system requirements are as follows:

- Java JDK 1.7 or later.
- Java Servlet Specification 2.3 or later.
- Web browser requirements:
  - Microsoft Internet Explorer 8.0 or later.
  - Mozilla Firefox 3.0 or later.
  - Google Chrome 2.0 or later.

**Note:** Oracle Application Express is *not* a prerequisite for using Oracle REST Data Services.

If Oracle Application Express is installed and if RESTful services have been configured during the installation (see the step "Configure RESTful Services" in Oracle Application Express Installation Guide), then Oracle REST Data Services supports it, including executing the RESTful services defined in Oracle Application Express.

1.2.3 About Installing Oracle REST Data Services

To install Oracle REST Data Services:

1. Download, install, and configure Oracle REST Data Services. See Section 1.3, "Installing and Configuring Oracle REST Data Services".
2. Deploy Oracle REST Data Services. Deployment options include:

■ **Standalone Mode.** See Section 1.4, "Running in Standalone Mode".

■ **Oracle WebLogic Server.** See Section 1.5, "Deploying to Oracle WebLogic Server".

■ **GlassFish Server.** See Section 1.6, "Deploying to GlassFish Server".

■ **Apache Tomcat.** See Section 1.7, "Deploying to Apache Tomcat".

**See Also:** Section 1.8, "Upgrading Oracle REST Data Services" and Appendix B, "Troubleshooting Oracle REST Data Services"

### 1.3 Installing and Configuring Oracle REST Data Services

Before you deploy Oracle REST Data Services, you must install and configure it using a command-line interface.

**NoSQL Database Users:** If you plan to use Oracle REST Data Services with a NoSQL Database store, see also Chapter 4, "NoSQL and Oracle REST Data Services", especially Section 4.2, "NoSQL Store Installation and Registration".

**Note:** To use the Oracle REST API for JSON Data Persistence, you must also install the Oracle REST API. See "Oracle REST API Installation" in Oracle REST Data Services SODA for REST Developer’s Guide.

**Topics:**

■ About Using the Command-Line Interface

■ About the Database Users Used by Oracle REST Data Services

■ Privileges Granted by Oracle REST Data Services

■ Procedures for Installing and Configuring Oracle REST Data Services

■ Using SQL Developer Oracle REST Data Services Administration (Optional)

■ Using OAuth2 in Non-HTTPS Environments

#### 1.3.1 About Using the Command-Line Interface

Oracle REST Data Services provides several command line commands. For example, you can configure the location where Oracle REST Data Services stores configuration files, configure the database Oracle REST Data Services uses, and start Oracle REST Data Services in standalone mode.

To display a full list of available commands, go to the directory or folder containing the ords.war file and execute the following command:

```
java -jar ords.war help
```

A list of the available commands is displayed. To see instructions on how to use each of these commands, enter help followed by the command name, for example:

```
java -jar ords.war help configdir
```
1.3.2 About the Database Users Used by Oracle REST Data Services

Oracle REST Data Services uses the following database users:

<table>
<thead>
<tr>
<th>User Name</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APEX_PUBLIC_USER</td>
<td>Only if using Oracle REST Data Services with Oracle Application Express</td>
<td>If you use Oracle REST Data Services with Oracle Application Express, this is the database user used when invoking PL/SQL Gateway operations, for example, all Oracle Application Express operations. For information on unlocking the APEX_PUBLIC_USER, see “Configure APEX_PUBLIC_USER Account” in Oracle Application Express Installation Guide.</td>
</tr>
<tr>
<td>APEX_REST_PUBLIC_USER</td>
<td>Only if using RESTful Services defined in Application Express</td>
<td>The database user used when invoking Oracle Application Express RESTful Services if RESTful Services defined in Application Express workspaces are being accessed.</td>
</tr>
<tr>
<td>APEX_LISTENER</td>
<td>Only if using RESTful Services defined in Application Express</td>
<td>The database user used to query RESTful Services definitions stored in Oracle Application Express if RESTful Services defined in Application Express workspaces are being accessed.</td>
</tr>
<tr>
<td>ORDS_METADATA</td>
<td>Yes</td>
<td>Owner of the PL/SQL packages used for implementing many Oracle REST Data Services capabilities. ORDS_METADATA is where the metadata about Oracle REST Data Services-enabled schemas is stored. It is not accessed directly by Oracle REST Data Services; the Oracle REST Data Services application never creates a connection to the ORDS_METADATA schema. The schema password is set to a random string, connect privilege is revoked, and the password is expired.</td>
</tr>
<tr>
<td>ORDS_PUBLIC_USER</td>
<td>Yes</td>
<td>User for invoking RESTful Services in the Oracle REST Data Services-enabled schemas.</td>
</tr>
</tbody>
</table>

The APEX_<xxx> users are created during the Oracle Application Express installation process.

1.3.3 Privileges Granted by Oracle REST Data Services

As part of the Oracle REST Data Services installation, privileges are granted to several users:

- **PUBLIC** is granted `SELECT` on many ORDS_METADATA tables and views.
- **PUBLIC** is granted `EXECUTE` on PL/SQL packages that are available for users to invoke.
- **ORDS_METADATA** is granted `EXECUTE` on the following:
  - SYS.DBMS_ASSERT
  - SYS.DBMS_CRYPTO
  - SYS.DBMS_LOB
SYS.DBS_OUTPUT
- SYS.DBS_REGISTRY
- SYS.DBS_SESSION
- SYS.DBSUTILITY
- SYS.VALIDATE_ORDS
- SYS.HTF
- SYS.HTP
- SYS.OWA
- SYS.WPG_DOCLOAD

ORDS_METADATA is granted SELECT on the following:
- SYS.DBA_DIRECTORIES
- SYS.DBA_OBJECTS

ORDS_METADATA is granted the following system privileges:
- ALTER USER
- CREATE TRIGGER

ORDS_METADATA is granted the necessary object privileges to migrate Application Express REST data to ORDS_METADATA tables.

1.3.4 Procedures for Installing and Configuring Oracle REST Data Services

The procedures in this topic apply to installing Oracle REST Data Services in a traditional (non-CDB) database. If you want to install and use Oracle REST Data Services in a multitenant database environment, see Appendix D, "Using the Multitenant Architecture with Oracle REST Data Services".

Note: You must complete the configuration steps in this topic before deploying to an application server.

To install and configure Oracle REST Data Services:

1. Download the file ords.version.number.zip from the Oracle REST Data Services download page. See:
   Note that the version.number in the file name reflects the current release number.

2. Unzip the downloaded zip file into a directory (or folder) of your choice:
   - UNIX and Linux: unzip ords.version.number.zip
   - Windows: Double-click the file ords.version.number.zip in Windows Explorer

3. Choose one of the following installation options:
   - Simple Installation Using a Parameter File
   - Advanced Installation Using Command-Line Prompts
   - Manual Installation (Mainly If You Do Not Have SYSDBA Privileges)
4. If you want to reinstall or uninstall Oracle REST Data Services, see Section 1.3.4.5, "If You Want to Reinstall or Uninstall (Remove) Oracle REST Data Services".

Related subtopics:

- Simple Installation Using a Parameter File
- Advanced Installation Using Command-Line Prompts
- Extracting the Scripts to Install and Uninstall Oracle REST Data Services
- Manual Installation (Mainly If You Do Not Have SYSDBA Privileges)
- If You Want to Reinstall or Uninstall (Remove) Oracle REST Data Services

See Also: Section 1.3.2, "About the Database Users Used by Oracle REST Data Services"

1.3.4.1 Simple Installation Using a Parameter File

You can perform a simple installation using the parameters specified in the params/ords_params.properties file under the location where you installed Oracle REST Data Services. You can edit that file beforehand to change default values to reflect your environment and preferences. If a parameter is missing in the file, you will be prompted for it.

To perform a simple installation using the parameter file, enter either of the following commands:

```
java -jar ords.war
java -jar ords.war install simple
```

(You can omit install simple because the simple installation is the default.)

The default ords_params.properties file includes the following:

```
db.hostname=localhost
db.port=1521
db.servicename=
db.sid=
db.username=APEX_PUBLIC_USER
migrate.apex.rest=false
rest.services.apex.add=
rest.services.ords.add=true
schema.tablespace.default=SYSAUX
schema.tablespace.temp=TEMP
standalone.http.port=8080
standalone.mode=true
standalone.static.images=
user.tablespace.default=USERS
user.tablespace.temp=TEMP
```

Windows Users: On Microsoft Windows systems, if you specify an Application Express static images location for standalone.static.images, use the backslash (escape character) before the colon, and use a forward slash as the folder separator. Example:

```
standalone.static.images=d:\:/test/apex426/apex/images/
```

When you install Oracle REST Data Services for the first time, you are prompted for passwords for these users: the PL/SQL gateway database user and ORDS_PUBLIC_
1.3.4.2 Advanced Installation Using Command-Line Prompts

You can perform an advanced installation in which you are prompted for the necessary parameter values, after which your choices are stored in the params/ords_params.properties file under the location where you installed Oracle REST Data Services.

To perform an advanced installation, enter the following command:

```
java -jar ords.war install advanced
```

During installation, Oracle REST Data Services checks if configuration files already exist in your specified configuration folder:

- If configuration files do not exist in that folder, they are created (examples: defaults.xml, apex_pu.xml).
- If configuration files from an earlier release exist in that folder, Oracle REST Data Services checks if <name>_pu.xml is present; and if it is not, you are prompted for the password for the ORDS_PUBLIC_USER account. If the configuration files <name>_al.xml and <name>_rt.xml from Release 2.0.n exist, they are preserved. (However, in Releases 2.0.n RESTful Services was optional, and therefore the files might not exist in the configuration folder.)
- If multiple configuration files exist from a previous release (examples: apex.xml, apex_al.xml, apex_rt.xml, sales.xml, sales_al.xml, sales_rt.xml, ...), and if <name>_pu.xml does not exist, then you are prompted to select the database configuration so that the ORDS schema can be created in that database.

The following shows an example advanced installation. In this example, if you accepted the default value of 1 for Enter 1 if you wish to start in standalone mode or 2 to exit [1], the remaining prompts are displayed; and if you will be using Oracle Application Express, you must specify the APEX static resources location.

```
d:\ords>java -jar ords.war install advanced
This Oracle REST Data Services instance has not yet been configured. Please complete the following prompts
Enter the location to store configuration data:d:\path\to\config\n Enter the name of the database server [localhost]:
 Enter the database listen port [1521]: Enter 1 to specify the database service name, or 2 to specify the database SID [1]:2
 Enter the database SID [xe]: Enter 1 if you want to verify/install Oracle REST Data Services schema or 2 to skip this step [1]:
 Enter the database password for ORDS_PUBLIC_USER: Confirm password:
 Please login with SYSDBA privileges to verify Oracle REST Data Services schema. Installation may be required.
 Enter the username with SYSDBA privileges to verify the installation [SYS]: Enter the database password for SYS: Confirm password:
Oracle REST Data Services schema does not exist and will be created.
```
Enter the default tablespace for ORDS_METADATA [SYSAUX]:
Enter the temporary tablespace for ORDS_METADATA [TEMP]:
Enter the default tablespace for ORDS_PUBLIC_USER [USERS]:
Enter the temporary tablespace for ORDS_PUBLIC_USER [TEMP]:
Enter 1 if you want to use PL/SQL Gateway or 2 to skip this step [1]:
Enter the PL/SQL Gateway database user name [APEX_PUBLIC_USER]:
Enter the database password for APEX_PUBLIC_USER:
Confirm password:
Enter 1 to specify passwords for Application Express RESTful Services database users (APEX_LISTENER, APEX_REST_PUBLIC_USER) or 2 to skip this step [1]:
Enter the database password for APEX_LISTENER:
Confirm password:
Enter the database password for APEX_REST_PUBLIC_USER:
Confirm password:
Enter 1 if you wish to start in standalone mode or 2 to exit [1]:
Enter the APEX static resources location:d:\apex426\apex\images\Enter the HTTP port [8080]:

1.3.4.3 Extracting the Scripts to Install and Uninstall Oracle REST Data Services

For convenience, you can use a single command to extract the scripts for installing and uninstalling Oracle REST Data Services. You can use an existing folder or one that will be created automatically for you. The command has the following format:

```
java -jar ords.war ords-scripts --scriptdir your-script-dir
```

Where:

- `--scriptdir` is an optional parameter for you to specify to store the extracted scripts.
- `your-script-dir` is the name of the folder to hold the extracted scripts. If this folder does not already exist, it is created.

1.3.4.4 Manual Installation (Mainly If You Do Not Have SYSDBA Privileges)

You can manually install Oracle REST Data Services by giving a SQL script to a user with SYSDBA privileges to execute it. The `ords_manual_install.sql` script creates the ORDS_SCHEMA and its database objects by executing several scripts. These scripts are provided in case the user does not have SYSDBA privileges, and the user can give the necessary scripts to the DBA to execute.

**Note:** To install Oracle REST Data Services into a CDB, you must perform a manual install operation, as explained in Appendix D, "Using the Multitenant Architecture with Oracle REST Data Services".

To perform a manual installation:

1. Extract scripts as explained in Section 1.3.4.3, "Extracting the Scripts to Install and Uninstall Oracle REST Data Services".
2. Start SQL*Plus and execute an appropriate statement, such as the following example

```
sqlplus "sys as sysdba" @ords_manual_install SYSAUX TEMP d:/log/scripts/
```

The `ords_manual_install.sql` script takes the following parameters:

- Default tablespace name for ORDS_METADATA schema (for example, SYSAUX)
- Temporary tablespace name for ORDS_METADATA schema (for example, TEMP)
1.3.4.5 If You Want to Reinstall or Uninstall (Remove) Oracle REST Data Services

If you want to reinstall Oracle REST Data Services, you must first uninstall the existing Oracle REST Data Services; and before you uninstall, ensure that Oracle REST Data Services is stopped.

Uninstalling Oracle REST Data Services removes the ORDS_METADATA schema, the ORDS_PUBLIC_USER user, and Oracle REST Data Services-related database objects (including public synonyms) if they exist in the database. To uninstall (remove, or deinstall) Oracle REST Data Services, go to the directory or folder containing the ords.war file and enter the `uninstall` command as follows:

```java -jar ords.war uninstall```

The `uninstall` command prompts you for some necessary information (host, port, SID or service name, username, password).

If you instead want to perform a **manual uninstallation**, follow these steps:

1. Extract the Oracle REST Data Services scripts, as explained in Section 1.3.4.3, "Extracting the Scripts to Install and Uninstall Oracle REST Data Services".
2. Go to the folder where you extracted the scripts.
3. Using SQL*Plus, connect as SYS AS SYSDBA.
4. Execute the `ords_manual_uninstall.sql` script. For example:

   ```sql
   @ords_manual_uninstall d:\scripts\logs
   ```

   Where the argument specifies the location for the Oracle REST Data Services log files (must already exist).

5. Verify that the uninstall operation was successful (a "completed successfully" message is displayed).

   If an error occurred, view the log files.

---

**Note:** To uninstall Oracle REST Data Services from a CDB, perform a manual uninstallation, as explained in Section D, "Using the Multitenant Architecture with Oracle REST Data Services".

---

1.3.5 Using SQL Developer Oracle REST Data Services Administration (Optional)

This section describes how to use Oracle SQL Developer to administer Oracle REST Data Services.

**See Also:** "Oracle REST Data Services Administration" in *Oracle SQL Developer User’s Guide*

**Topics:**

- About SQL Developer Oracle REST Data Services Administration
- Configuring an Administrator User
1.3.5.1 About SQL Developer Oracle REST Data Services Administration

Oracle SQL Developer enables you to administer Oracle REST Data Services using a graphical user interface. To take full advantage of these administration capabilities, you must use SQL Developer Release 4.1 or later. Using SQL Developer for Oracle REST Data Services administration is optional.

Using this graphical user interface, you can update the database connections, JDBC settings, URL mappings, RESTful connections, security (allowed procedures, blocked procedures, validation function and virus scanning), Caching, Pre/Post Processing Procedures, Environment, and Excel Settings. Oracle SQL Developer also provides statistical reporting, error reporting, and logging.

See Also: "Oracle REST Data Services Administration" in Oracle SQL Developer User’s Guide

1.3.5.2 Configuring an Administrator User

If you want to be able to administer Oracle REST Data Services using SQL Developer, then you must configure an administrator user as follows:

- Execute the following command:
  ```bash
  java -jar ords.war user adminlistener "Listener Administrator"
  ```
- Enter a password for the `adminlistener` user.
- Confirm the password for the `adminlistener` user.
- If you are using Oracle REST Data Services without HTTPS, follow the steps in Section 1.3.6, "Using OAuth2 in Non-HTTPS Environments".

When using SQL Developer to retrieve and/or upload an Oracle REST Data Services configuration, when prompted, enter the credentials provided in the preceding list.

1.3.6 Using OAuth2 in Non-HTTPS Environments

RESTful Services can be protected with the OAuth2 protocol to control access to nonpublic data. To prevent data snooping, OAuth2 requires all requests involved in the OAuth2 authentication process to be transported using HTTPS. The default behavior of Oracle REST Data Services is to verify that all OAuth2 related requests have been received using HTTPS. It will refuse to service any such requests received over HTTP, returning an HTTP status code of 403 Forbidden.

This default behavior can be disabled in environments where HTTPS is not available as follows:

1. Locate the folder where the Oracle REST Data Services configuration is stored.
2. Edit the file named `defaults.xml`.
3. Add the following setting to the end of this file just before the `</properties>` tag.
   ```xml
   <entry key="security.verifySSL">false</entry>
   ```
4. Save the file.
5. Restart Oracle REST Data Services if it is running.

Note that it is only appropriate to use this setting in development or test environments. It is never appropriate to use this setting in production environments because it will result in user credentials being passed in clear text.
1.4 Running in Standalone Mode

Although Oracle REST Data Services supports the Java EE application servers described in Section 1.2.1, "Supported Java EE Application Servers", you also have the option of running in standalone mode. This section describes how to run Oracle REST Data Services in standalone mode.

Standalone mode is suitable for development use only, and is not supported for use in production deployments. Use a supported Java EE application server for production deployments.

Topics:
- Starting in Standalone Mode
- Stopping the Server in Standalone Mode
- Configuring a Doc Root for Non-Application Express Static Resources

1.4.1 Starting in Standalone Mode

To launch Oracle REST Data Services in standalone mode:

1. To start Standalone mode, execute the following command:
   
   `java -jar ords.war`

   If you have not yet completed configuration, you are prompted to do so.

   **Tip:** To see help on standalone mode options, execute the following command:

   `java -jar ords.war help standalone`

   **Note:** Standalone mode does not support HTTPS, so the steps in Section 1.3.6, "Using OAuth2 in Non-HTTPS Environments" must be followed if you want to use RESTful Services that require secure access (including the SQL Developer Administration View).

2. When prompted, specify the location of the folder containing the Oracle Application Express static resources used by Oracle REST Data Services, or press Enter if you do not want to specify this location.

3. When prompted, enter the value of the HTTP port you want the standalone Oracle REST Data Services to listen on. The default port is 8080.

   You are only prompted for these values the first time you launch standalone mode.

   **Note:** Ensure that no other servers are listening on the port you choose. The default port 8080 is commonly used by HTTP or application servers including the embedded PL/SQL gateway.
1.4.2 Stopping the Server in Standalone Mode

To stop the Oracle REST Data Services server in standalone mode, at a command prompt press Ctrl+C.

1.4.3 Configuring a Doc Root for Non-Application Express Static Resources

You can configure a doc root for standalone mode to deploy static resources that are outside the /i folder that is reserved for Application Express static resources.

To do so, specify the --doc-root parameter with the standalone mode command, as in the following example:

```
java -jar ords.war standalone --doc-root /var/www/html
```

The preceding example makes any resource located within /var/www/html available under http://server:port/. For example, if the file /var/www/html/hello.txt exists, it will be accessible at http://server:port/hello.txt.

The value specified for --doc-root is stored in

```
${config.dir}/ords/standalone/standalone.properties
```

in the standalone.doc.root property. If a custom doc root is not specified using --doc-root, then the default doc-root value of ${config.dir}/ords/standalone/doc_root is used. Any file placed within this folder will be available at the root context.

This approach has the following features and considerations:

- HTML resources can be addressed without their file extension. For example, if a file named hello.html exists in the doc root, it can be accessed at the URI http://server:port/hello.
- Attempts to address a HTML resource with its file extension are redirected to the location without an extension. For example, if the URI http://server:port/hello.html is accessed, then the client is redirected to http://server:port/hello.
- The usual practice is to serve HTML resources without their file extensions, so this feature facilitates that practice, while the redirect handles the case where the resource is addressed with its file extension.
- Index pages for folders are supported. If a folder contains a file named index.html or index.htm, then that file is used as the index page for the folder. For example, if /var/www/html contains /abc/xyz/index.html, then accessing http://server:port/abc/xyz/ displays the contents of index.html.
- Addressing a folder without a trailing slash causes a redirect to the URI with a trailing slash. For example, if a client accesses http://server:port/abc/xyz, then the server issues a redirect to http://server:port/abc/xyz/.
- Resources are generated with weak etags based on the modification stamp of the file and with a Cache Control header that causes the resources to be cached for 1 hour.

1.5 Deploying to Oracle WebLogic Server

This section describes how to deploy Oracle REST Data Services on Oracle WebLogic Server. It assumes that you have completed the installation process and are familiar with Oracle WebLogic Server. If you are unfamiliar with domains, managed servers, deployment, security, users and roles, refer to your Oracle WebLogic Server documentation.
Deploying to Oracle WebLogic Server

1.5 About Oracle WebLogic Server


To learn more about installing Oracle WebLogic Server, see Oracle Fusion Middleware Getting Started With Installation for Oracle WebLogic Server and Oracle Fusion Middleware Installation Guide for Oracle WebLogic Server.

1.5.2 Downloading, Installing, and Configuring Oracle REST Data Services

You must complete this step before deploying Oracle REST Data Services on WebLogic. For more information, see Section 1.3, "Installing and Configuring Oracle REST Data Services".

1.5.3 Configuring Oracle Application Express Images

If you are using Oracle Application Express, you must create a web archive to reference the Oracle Application Express, image files. However, if you are not using Oracle Application Express, you may skip the rest of this section about configuring Oracle Application Express images.

Before you begin, you must create a web archive (WAR) file to reference the Oracle Application Express image files. Use the static command to create a web archive file named i.war:

```
java -jar ords.war static <apex directory>/images
```

Where:

- `<apex directory>` is the directory location of Oracle Application Express.

This command runs the `static` command contained in the `ords.war` file. It packages the Application Express static images into an archive file named `i.war`.

The created images WAR does not contain the static resources; instead, it references the location where the static resources are stored. Therefore the static resources must be available at the specified path on the server where the WAR is deployed.

**Tip:** Use `java -jar ords.war help static` to see the full range of options for the `static` command.

Use the `i.war` file to deploy to WebLogic in the following steps:

1. Launching the Administration Server Console
2. Installing the Oracle WebLogic Server Deployment
3. Configuring WebLogic to Handle HTTP Basic Challenges Correctly

1.5.4 Launching the Administration Server Console

To launch the Administration Server console:

1. Start an Administration Server.
2. Launch the WebLogic Administration Console by typing the following URL in your web browser:
   
   \[http://<host>:<port>/console\]

   Where:
   - `<host>` is the DNS name or IP address of the Administration Server.
   - `<port>` is the port on which the Administration Server is listening for requests (port 7001 by default).
3. Enter your WebLogic Administrator username and password.
4. If your domain is in `Production` mode, click the `Lock & Edit` button on the left-pane below the submenu Change Center. If your domain is in `Development` mode, this button does not appear.

1.5.5 Installing the Oracle WebLogic Server Deployment

**Tip:** The Oracle REST Data Services files, `ords.war` and `i.war`, must be available before you start this task. See Section 1.3, "Installing and Configuring Oracle REST Data Services" and Section 1.5.3, "Configuring Oracle Application Express Images"

To install the deployment:

   
   The Summary of Deployments is displayed.
2. Click Install.
3. Specify the location of the `ords.war` file and click Next.

   The `ords.war` file is located in the folder where you unzipped the Oracle REST Data Services ZIP file. See Section 1.3, "Installing and Configuring Oracle REST Data Services".

   **Tip:** WebLogic Server determines the context root from the file name of a WAR archive. If you need to keep backward compatibility, so that URLs are of the form `http://server/apex/...` rather than `http://server/ords/...`, then you must rename `ords.war` to `apex.war` before the deployment.

   The Install Application assistant is displayed.
4. Select Install this deployment as an application and click Next.
5. Select the servers and/or clusters to which you want to deploy the application or module and click Next.
Tip: If you have not created additional Managed Servers or clusters, you do not see this assistant page.

6. In the Optional Settings, specify the following:
   a. Name - Enter: ords
   b. Security - Select the following:
      Custom Roles: Use roles that are defined in the Administration Console; use policies that are defined in the deployment descriptor
   c. Source accessibility - Select:
      Use the defaults defined by the deployment's targets

7. Click Next.
   A summary page is displayed.

8. Under Additional configuration, select one of the following:
   ■ Yes, take me to the deployment's configuration - Displays the Configuration page.
   ■ No I will review the configuration later - Returns you to the Summary of Deployments page.

9. Review the summary of configuration settings that you have specified.

10. Click Finish.

11. Repeat the previous steps to deploy the i.war file.
    In the optional settings, specify the following:
    a. Name - Enter: i
    b. Security - Select:
       Custom Roles: Use roles that are defined in the Administration Console; use policies that are defined in the deployment descriptor
    c. Source Accessibility - Select:
       Use the defaults defined by the deployment's targets

12. If your domain is in Production Mode, then on the Change Center click Activate Changes.

1.5.6 Configuring WebLogic to Handle HTTP Basic Challenges Correctly

By default WebLogic attempts to intercept all HTTP Basic Authentication challenges. This default behavior needs to be disabled for Oracle REST Data Services to function correctly.

See your WebLogic documentation for the location of the WebLogic configuration file named: config.xml

Add the <enforce-valid-basic-auth-credentials> element to config.xml within the<security-configuration> element. The edited file should look like the following:

...
<enforce-valid-basic-auth-credentials>false</enforce-valid-basic-auth-credentials>
</security-configuration>

Save the updated config.xml file, and restart WebLogic if it is running.

### 1.5.7 Verifying the State and Health of ords and i

In the Summary of Deployments, select the Control tab and verify that both the ords and i State are Active and the Health status is OK.

If ords and/or i are not Active, then enable them. In the Deployments table, select the check box next to ords and/or i. Click Start and select Servicing all requests to make them active.

### 1.6 Deploying to GlassFish Server

This section describes how to deploy Oracle REST Data Services on GlassFish Server.

**Topics:**
- About GlassFish Server
- Downloading, Installing, and Configuring Oracle REST Data Services
- Configuring Oracle Application Express Images
- Launching the Administration Server Console
- Installing the GlassFish Server Deployment

**Tip:** This section assumes that you have completed the installation process and are familiar with GlassFish Server. If you are unfamiliar with domains, servers, applications, security, users and roles, see your GlassFish Server documentation.

#### 1.6.1 About GlassFish Server

You can install Oracle REST Data Services with GlassFish Server. GlassFish Server is available for download from the Oracle Technology Network. See:


#### 1.6.2 Downloading, Installing, and Configuring Oracle REST Data Services

You must complete this step before deploying Oracle REST Data Services on GlassFish. For more information, see Section 1.3, "Installing and Configuring Oracle REST Data Services".

#### 1.6.3 Configuring Oracle Application Express Images

If you are using Oracle Application Express, you must create a web archive to reference the Oracle Application Express, image files. However, if you are not using Oracle Application Express, you may skip the rest of this section about configuring Oracle Application Express images.

Before you begin, you must create a web archive (WAR) file to reference the Oracle Application Express image files. Use the `static` command to create a web archive file named `i.war:`
Deploying to GlassFish Server

Installing Oracle REST Data Services

```
java -jar ords.war static <apex directory>
```

Where:

- `<apex directory>` is the directory location of Oracle Application Express.

The created images WAR does not contain the static resources; instead, it references the location where the static resources are stored. Therefore the static resources must be available at the specified path on the server where the WAR is deployed.

**Tip:** Use `java -jar ords.war help static` to see the full range of options for the static command.

Use the `i.war` file to deploy to GlassFish in the following steps:

1. **Launching the Administration Server Console**
2. **Installing the GlassFish Server Deployment**

### 1.6.4 Launching the Administration Server Console

At least one GlassFish server domain must be started before you start the Administration Console.

To launch the Administration Console:

1. Launch the Administration Console by typing the following URL in your web browser:
   ```
   http://localhost:4848
   ```

2. If prompted, log in to the Administration Console.

   **Tip:** You are prompted to log in if you chose to require an administration password at the time GlassFish server was installed.

### 1.6.5 Installing the GlassFish Server Deployment

**Tip:** The Oracle REST Data Services files, `ords.war` and `i.war` must be available before you start this task. See Section 1.3, "Installing and Configuring Oracle REST Data Services" and Section 1.6.3, "Configuring Oracle Application Express Images".

To install the deployment:

1. On the navigation tree, click the **Application** node.
   The Applications page is displayed.

2. Click the **Deploy** button.
   The Deploy Applications or Modules page is displayed.

3. Select **Packaged File to be Uploaded to the Server** and click **Browse**.

4. Navigate to the location of the `ords.war` file, select the file, and click **Open**.
   The Deploy Applications or Modules page is displayed.

5. On the Deploy Applications or Modules page, specify the following:
   a. **Type**: Web Application
   b. **Context Root**: `ords`
Tip: The Context Root value defaults to ords. However you can change it to apex if you need to keep backward compatibility, so that URLs are of the form http://server/apex/... rather than http://server/ords/....

c. Application Name: ords
d. Status: Enabled
e. Description: Oracle REST Data Services
f. Accept all other default settings and click OK.

6. Repeat the previous steps to deploy the i.war file. Clear the Context Root field so that the context root set in the sun-web.xml is used.

The Applications page is displayed. A check mark should appear in the Enabled field for ords

Tip: If a check mark does not appear in the Enabled column for ords, then select the check box next to ords and click Enable.

1.7 Deploying to Apache Tomcat

This section describes how to deploy Oracle REST Data Services on Apache Tomcat.

Topics:
- About Apache Tomcat
- Downloading, Installing, and Configuring Oracle REST Data Services
- Configuring Oracle Application Express Images
- Installing the Apache Tomcat Deployment

1.7.1 About Apache Tomcat

You can download Apache Tomcat from:

http://tomcat.apache.org/download-70.cgi

Tip: This section assumes that you have completed the installation process and are familiar with Apache Tomcat. If you are unfamiliar with domains, servers, applications, security, users and roles, see your Apache Tomcat documentation.

1.7.2 Downloading, Installing, and Configuring Oracle REST Data Services

You must complete this step before deploying Oracle REST Data Services on Apache Tomcat. For more information, see Section 1.3, "Installing and Configuring Oracle REST Data Services".

1.7.3 Configuring Oracle Application Express Images

If you are using Oracle Application Express, you must create a web archive to reference the Oracle Application Express, image files. However, if you are not using Oracle Application Express, you may skip the rest of this section about configuring Oracle Application Express images.

To configure Oracle Application Express Images on Apache Tomcat:
Copy the contents of the `<apex directory>/images` folder to `<Tomcat directory>/webapps/i/`.

Where:

- `<apex directory>` is the directory location of the Oracle Application Express distribution.
- `<Tomcat directory>` is the folder where Apache Tomcat is installed.

### 1.7.4 Installing the Apache Tomcat Deployment

**Tip:** The Oracle REST Data Services file `ords.war` must be available before you start this task. See Section 1.3, "Installing and Configuring Oracle REST Data Services" and Section 1.7.3, "Configuring Oracle Application Express Images".

To install the Apache Tomcat deployment:

1. Move the `ords.war` file into the `webapps` folder where Apache Tomcat is installed.

   **Tip:** Apache Tomcat determines the context root from the file name of a WAR archive. If you need to keep backward compatibility, so that URLs are of the form `http://server/apex/...` rather than `http://server/ords/...`, then you must rename `ords.war` to `apex.war` before moving it into to the `webapps` folder.

2. Access Oracle Application Express by typing the following URL in your web browser:

   ```
   http://<hostname>:<port>/ords/
   ```

   Where:

   - `<hostname>` is the name of the server where Apache Tomcat is running.
   - `<port>` is the port number configured for Apache Tomcat application server.

### 1.8 Upgrading Oracle REST Data Services

In Oracle REST Data Services 3.0 and later, the configuration files include some additional options. For RESTful Services (also known as Resource Templates), the preferred storage is in the Oracle REST Data Services schema, although storing them in the Oracle Application Express schema is also supported.

If the configuration folder specified during Section 1.3, "Installing and Configuring Oracle REST Data Services" contains an existing Oracle REST Data Services 2.0 configuration, migration is not necessary; however, an option is provided to migrate RESTful Services out of the Oracle Application Express schema into the Oracle REST Data Services schema.
2 Configuring Oracle REST Data Services (Advanced)

This section explains how to configure Oracle REST Data Services for connecting to multiple databases for routing requests, and it refers to other documentation sources for other configuration information. (It supplements the basic configuration information in Section 1.3, "Installing and Configuring Oracle REST Data Services").

**Note:** Oracle REST Data Services must be restarted after making configuration changes. See your application server documentation for information on how to restart applications.

**Topics:**
- Configuring Multiple Databases
- Configuring Security, Caching, Pre- and Post Processing, Environment, and Excel Settings
- Developing RESTful Services for Use with Oracle REST Data Services

### 2.1 Configuring Multiple Databases

Oracle REST Data Services supports the ability to connect to more than one database. This section describes different strategies for routing requests to the appropriate database.

**Topics:**
- About the Request URL
- Configuring Additional Databases
- Routing Based on the Request Path Prefix
- Routing Based on the Request URL Prefix

#### 2.1.1 About the Request URL

Oracle REST Data Services supports a number of different strategies for routing requests to the appropriate database. All of these strategies rely on examining the request URL and choosing the database based on some kind of match against the URL. It is useful to recap the pertinent portions of a request URL. Consider the following URL:

https://www.example.com/ords/sales/f?p=1:1
This URL consists of the following sections:

- **Protocol:** https
- **Host Name:** www.example.com
- **Context Root:** /ords

The context root is the location at which Oracle REST Data Services is deployed on the application server.

- **Request Path:** /sales/f?p=1.1

This is the portion of the request URL relative to the context root.

For different applications, it may be important to route requests based on certain prefixes in the request path or certain prefixes in the full request URL.

There are two steps to configuring multiple databases:

1. Configuring the database connection information
2. Configuring which requests are routed to which database

### 2.1.2 Configuring Additional Databases

When you first configure Oracle REST Data Services, you configure a default database connection named: apex. You can create additional database connections using the `setup` command.

**Tip:** To see full help for the `setup` command type:

```
java -jar ords.war help setup
```

To create a database connection type the following:

```
java -jar ords.war setup --database <database name>
```

Where:

- `<database name>` is the name you want to give the database connection.

You are prompted to enter the information required to configure the database. See Section 1.3, "Installing and Configuring Oracle REST Data Services" for more information on the data that must be entered.

After you have configured the additional databases, define the rules for how requests are routed to the appropriate database: see Section 2.1.3, "Routing Based on the Request Path Prefix" and Section 2.1.4, "Routing Based on the Request URL Prefix".

### 2.1.3 Routing Based on the Request Path Prefix

You create request routing rules using the `map-url` command.

**Tip:** To see full help for the `map-url` command type:

```
java -jar ords.war help map-url
```

If you want to route requests based just on matching a prefix in the request path portion of the URL, use the `map-url` command as follows:

```
java -jar ords.war map-url --type base-path --workspace-id <workspace name> <path
```
prefix> <database name>

Where:

- `<workspace name>` is the name of the Oracle Application Express workspace where RESTful services for this connection are defined. This may be omitted if RESTful Services are not being used.
- `<path prefix>` is the prefix that must occur at the start of the request path.
- `<database name>` is the name of the database connection configured in the previous step, Section 2.1.2, ”Configuring Additional Databases”.

### 2.1.3.1 Example of Routing Based on the Request Path Prefix

Assuming Oracle REST Data Services is deployed on a system named example.com at the context path /ords, then create the following rule:

```
java -jar ords.war map-url --type base-path --workspace-id sales_rest /sales sales_db
```

This rule means that any requests matching [https://example.com/ords/sales/...](https://example.com/ords/sales/...) are routed to the `sales_db` database connection. The `sales_rest` workspace defined within the `sales_db` database is searched for RESTful Services definitions.

The previous rule matches all of the following requests:

- [https://example.com/ords/sales/f?p=1:1](https://example.com/ords/sales/f?p=1:1)
- [https://example.com/ords/sales/leads/](https://example.com/ords/sales/leads/)
- [https://www.example.com/ords/sales/forecasting.report?month=jan](https://www.example.com/ords/sales/forecasting.report?month=jan)  (If www.example.com resolves to the same system as example.com.)

The previous rule does not match any of the following requests:

- [https://example.com:8080/ords/sales/f?p=1:1](https://example.com:8080/ords/sales/f?p=1:1)  (The port is wrong: 443 is default for https, but don’t specify if using default.)
- [https://example.com/ords/f?p=1:1](https://example.com/ords/f?p=1:1)  (Missing the /sales prefix.)
- [https://example.com/pls/sales/leads/](https://example.com/pls/sales/leads/)  (The context path is wrong.)

### 2.1.4 Routing Based on the Request URL Prefix

If you want to route requests based on a match of the request URL prefix, use the `map-url` command as follows:

```
java -jar ords.war map-url --type base-url --workspace-id <workspace name> <url prefix> <database name>
```

Where:

- `<workspace name>` is the name of the Oracle Application Express workspace where RESTful services for this connection are defined. This may be omitted if RESTful Services are not being used.
- `<url prefix>` is the prefix with which the request URL must start.
- `<database name>` is the name of the database connection.

### 2.1.4.1 Example of Routing Based on the Request URL Prefix

Assuming Oracle REST Data Services is deployed on a system named example.com at the context path /ords, then create the following rule:

```
java -jar ords.war map-url --type base-url --workspace-id sales_rest
```
https://example.com/ords/sales sales_db

This rule means that any requests matching https://example.com/ords/sales/... are routed to the sales_db database connection. The sales_rest workspace defined within the sales_db database is searched for RESTful Services definitions.

The previous rule matches all of the following requests:

https://example.com/ords/sales/f?p=1:1
https://example.com/ords/sales/leads/
https://example.com/ords/sales/forecasting.report?month=jan

The previous rule does not match any of the following requests:

http://example.com/ords/sales/f?p=1:1 (The protocol is wrong.)
https://example.com:8080/ords/sales/f?p=1:1 (The port is wrong: 443 is default for https, but don’t specify if using default.)
https://example.com/ords/f?p=1:1 (Missing the /sales segment of the base URL.)
https://example.com/pls/sales/leads/ (The context path is wrong.)
https://www.example.com/ords/sales/forecasting.report?month=jan (The host name is wrong.)

2.2 Configuring Security, Caching, Pre- and Post Processing, Environment, and Excel Settings

To configure security, caching, pre- and post-processing, environment, and Excel settings, see Section 1.3.5, “Using SQL Developer Oracle REST Data Services Administration (Optional)”.

2.3 Developing RESTful Services for Use with Oracle REST Data Services

For more information on how to develop RESTful Services for use with Oracle REST Data Services, see Chapter 3, "Developing Oracle REST Data Services Applications".
Developing Oracle REST Data Services Applications

This section explains how to develop applications that use Oracle REST Data Services. It includes guidance and examples.

"Getting Started" Tutorial: If you want to get started quickly, you can try the tutorial in Appendix E, "Getting Started with RESTful Services" now. However, you should then return to this chapter to understand the main concepts and techniques.

Note: Ensure that you have installed and configured both Oracle Application Express 4.2 or later, and Oracle REST Data Services 3.0 or later, before attempting to follow any of the tutorials and examples.

To use the Oracle REST API for JSON Data Persistence, you must first install the Oracle REST API. See "Oracle REST API Installation" in Oracle REST Data Services SODA for REST Developer’s Guide.

It is assumed that you are familiar with Oracle Application Express. If you are new to Oracle Application Express, see the Oracle Application Express documentation.

Topics:
- Introduction to Relevant Software
- Getting Started with RESTful Services
- Automatic Enabling of Schema Objects for REST Access (AutoREST)
- Filtering in Queries
- Configuring Secure Access to RESTful Services
- About Oracle REST Data Services User Roles
- Authenticating Against WebLogic Server and GlassFish User Repositories
- Integrating with Existing Group/Role Models
- Using the Oracle REST Data Services PL/SQL API

You may also want to review Appendix C, “Development Tutorial: Creating an Image Gallery”, a supplementary extended example that uses Oracle Application Express to build an application.


3.1 Introduction to Relevant Software

This section explains some key relevant software for developing applications that use Oracle REST Data Services. (See also Section 1.1, "About Oracle REST Data Services").

Topics:

- About Oracle Application Express
- About RESTful Web Services

3.1.1 About Oracle Application Express

Oracle Application Express is a declarative, rapid web application development tool for the Oracle database. It is a fully supported, no cost option available with all editions of the Oracle database. Using only a web browser, you can develop and deploy professional applications that are both fast and secure.

3.1.2 About RESTful Web Services

Representational State Transfer (REST) is a style of software architecture for distributed hypermedia systems such as the World Wide Web. An API is described as RESTful when it conforms to the tenets of REST. Although a full discussion of REST is outside the scope of this document, a RESTful API has the following characteristics:

- Data is modelled as a set of resources. Resources are identified by URIs.
- A small, uniform set of operations are used to manipulate resources (for example, PUT, POST, GET, DELETE).
- A resource can have multiple representations (for example, a blog might have an HTML representation and an RSS representation).
- Services are stateless and since it is likely that the client will want to access related resources, these should be identified in the representation returned, typically by providing hypertext links.

Release 4.2 of Oracle Application Express leverages the capabilities of Oracle REST Data Services to provide developers with an easy to use graphical user interface for defining and testing RESTful Web Services.

3.2 Getting Started with RESTful Services

This section introduces RESTful Services, and provides guidelines and examples for developing applications that use RESTful Services.

Topics:

- RESTful Services Terminology
- Accessing RESTful Services Using Application Express
- "Getting Started" Documents Included in Installation
- Exploring the Sample RESTful Services (Tutorial)
- About cURL and Testing RESTful Services

3.2.1 RESTful Services Terminology

This section introduces some common terms that are used throughout this document:
RESTful service: An HTTP web service that conforms to the tenets of the RESTful architectural style described in About RESTful Web Services.

Resource module: An organizational unit that is used to group related resource templates.

Resource template: An individual RESTful service that is able to service requests for some set of URIs (Universal Resource Identifiers). The set of URIs is defined by the URI Pattern of the Resource Template

URI pattern: A pattern for the resource template. Can be either a route pattern or a URI template, although you are encouraged to use route patterns.

Route pattern: A pattern that focuses on decomposing the path portion of a URI into its component parts. For example, a pattern of /:object/:id? will match /emp/101 (matches a request for the item in the emp resource with id of 101) and will also match /emp/ (matches a request for the emp resource, because the :id parameter is annotated with the ? modifier, which indicates that the id parameter is optional).

For a detailed explanation of route patterns, see docs/javadoc/plugin-api/route-patterns.html, under <sqldeveloper-install>/ords and under the location (if any) where you manually installed Oracle REST Data Services.

URI template: A simple grammar that defines the specific patterns of URIs that a given resource template can handle. For example, the pattern employees/{id} will match any URI whose path begins with employees/, such as employees/2560.

Resource handler: Provides the logic required to service a specific HTTP method for a specific resource template. For example, the logic of the GET HTTP method for the preceding resource template might be:

```
select empno, ename, dept from emp where empno = :id
```

HTTP operation: HTTP (HyperText Transport Protocol) defines standard methods that can be performed on resources: GET (retrieve the resource contents), POST (store a new resource), PUT (update an existing resource), and DELETE (remove a resource).

### 3.2.2 Accessing RESTful Services Using Application Express

If you want to view, create, edit and test RESTful Services in Oracle Application Express, you can access the RESTful Services option within the Oracle Application Express SQL Workshop:

1. Log in to Oracle Application Express as a Developer or as an Administrator.
2. On the page displayed after login, click the icon labeled SQL Workshop.
3. On the next page, click the icon labeled RESTful Services.

However, the use of Oracle Application Express to define RESTful Services is supported for backwards compatibility, but you are encouraged to use Oracle REST Data Services-enabled schemas and SQL Developer instead.

Oracle REST Data Services-enabled schemas provide several benefits:

- Improved JSON generation that is consistent with other Oracle products
- Support for OAuth 2.0 Client credentials flow
- First Party cookie authentication
AutoREST (Automatically enabling schemas and objects for Oracle REST Data Services access)

No additional new features are planned for Oracle Application Express RESTful services. You can migrate an Application Express workspace to an Oracle REST Data Services-enabled schema; however, after you do that, you cannot use Application Express to create and edit RESTful services, but must instead use SQL Developer or the Oracle REST Data Services PL/SQL API.

3.2.3 "Getting Started" Documents Included in Installation

When you install Oracle REST Data Services, an examples folder is created with subfolders and files that you may find helpful. The installation folder hierarchy includes this:

```
ords
 conf
   docs
   examples
      soda
      getting-started
      getting-started-nosql
...
```

In this hierarchy:

- **examples\soda**: Contains sample JSON documents used in some examples included in Oracle REST Data Services SODA for REST Developer’s Guide.

- **examples\getting-started**: Double-click index.html for a short document about how to get started developing RESTful Services using Oracle REST Data Services. This document focuses on using SQL Developer to get started. (SQL Developer is the primary tool for managing Oracle REST Data Services. For example, the ability to auto-enable REST support for schemas and tables is available only in SQL Developer.)

- **examples\getting-started-nosql**: Double-click index.html for a short document about how to get started accessing NoSQL stores using Oracle REST Data Services.

3.2.4 Exploring the Sample RESTful Services (Tutorial)

If your Application Express instance is configured to automatically add the sample application and sample database objects to workspaces, then a sample resource module named: `oracle.example.hr` will be visible in the list of Resource Modules. If that resource module is not listed, then you can click the Reset Sample Data task on the right side of the RESTful Services Page to create the sample resource module.

1. Click on `oracle.example.hr` to view the Resource Templates and Resource Handlers defined within the module. Note how the module has a URI prefix with the value: `hr/`. This means that all URIs serviced by this module will start with the characters `hr/`.

2. Click on the resource template named `employees/{id}`. Note how the template has a URI Template with the value: `employees/{id}`. This means that all URIs starting with `hr/employees/` will be serviced by this Resource Template. The HTTP methods supported by a resource template are listed under the resource template. In this case, the only supported method is the GET method.

3. Click on the GET Resource Handler for `hr/employees/{id}` to view its configuration.
The **Source Type** for this handler is **Query One Row**. This means that the resource is expected to be mapped to a single row in the query result set. The Source for this handler is:

```sql
select * from emp
    where empno = :id
```

Assuming that the `empno` column is unique, the query should only produce a single result (or no result at all if no match is found for `:id`). To try it out, press the **Test** button. The following error message should be displayed:

400 - Bad Request - Request path contains unbound parameters: id

If you look at the URI displayed in the browser, it will look something like this:

https://server:port/ords/workspace/hr/employees/{id}

where:

- **server** is the DNS name of the server where Oracle Application Express is deployed
- **port** is the port the server is listening on
- **workspace** is the name of the Oracle Application Express workspace you are logged into

Note the final part of the URI: `hr/employees/{id}`. The error message says that this is not a valid URI, the problem is that you did not substitute in a concrete value for the parameter named `{id}`. To fix that, press the browser **Back** button, then click **Set Bind Variables**.

4. For the bind variable named `:id`, enter the value 7369, and press **Test**.

A new browser window appears displaying the following JSON (JavaScript Object Notation):

```json
{
    "empno":7369,
    "ename":"SMITH",
    "job":"CLERK",
    "mgr":7902,
    "hiredate":"1980-12-17T08:00:00Z",
    "sal":800,
    "deptno":20
}
```

Note also the URI displayed in the browser for this resource:

https://server:port/ords/workspace/hr/employees/7369

The `{id}` **URI Template parameter** is bound to the SQL `:id` bind variable, and in this case it has been given the concrete value of 7369, so the query executed by the **RESTful Service** becomes:

```sql
select * from emp
    where empno = 7369
```

The results of this query are then rendered as JSON as shown above.
Tip: Reading JSON can be difficult. To make it easier to read, install a browser extension that pretty prints the JSON. For example, Mozilla Firefox and Google Chrome both have extensions named JSONView:

- JSONView for Firefox
- JSONView for Chrome

Now see what happens when you enter the URI of a resource that does not exist.

5. On the Set Bind Variables page, change the value of :id from 7369 to 1111, and press Test.

As before, a new window pops up, but instead of displaying a JSON resource, it displays an error message reading:

404 - Not Found

This is the expected behavior of this handler: when a value is bound to :id that does not exist in the emp table, the query produces no results and consequently the standard HTTP Status Code of 404 - Not Found is returned.

So, you have a service that will provide information about individual employees, if you know the ID of an employee, but how do you discover the set of valid employee ids?

6. Press Cancel to return to the previous page displaying the contents of the Resource Module.

7. Click on the template named employees/.

The following steps look at the resource it generates, and later text will help you understand its logic.

8. Click on the GET handler beneath employees/, and click Test.

A resource similar to the following is displayed (If you haven’t already done so, now would be a good time to install a JSON viewer extension in your browser to make it easier to view the output):

```json
{
  "next":
  {
    "$ref":
    "https://server:port/ords/workspace/hr/employees/?page=1"},
  "items": [
  {
    "uri":
    {
      "$ref":
      "https://server:port/ords/workspace/hr/employees/7369"),
    "empno": 7369,
    "ename": "SMITH"
  },
  {
    "uri":
    {
      "$ref":
      "https://server:port/ords/workspace/hr/employees/7499"),
    "empno": 7499,
    "ename": "ALLEN"
  }...
  {
    "uri":
    {
      "$ref":
```
This JSON document contains a number of things worth noting:

- The first element in the document is named `next` and is a URI pointing to the next page of results. (An explanation of how paginated results are supported appears in later steps)

- The second element is named `items` and contains a number of child elements. Each child element corresponds to a row in the result set generated by the query.

- The first element of each child element is named `uri` and contains a URI pointing to the service that provides details of each employee. Note how the latter part of the URI matches the URI Template: `employees/{id}`. In other words, if a client accesses any of these URIs, the request will be serviced by the `employees/{id}` RESTful service previously discussed.

So, this service addresses the problem of identifying valid employee IDs by generating a resource that lists all valid employee resources. The key thing to realize here is that it does not do this by just listing the ID value by itself and expecting the client to be able to take the ID and combine it with prior knowledge of the `employees/{id}` service to produce an employee URI; instead, it lists the URIs of each employee.

Because the list of valid employees may be large, the service also breaks the list into smaller pages, and again uses a URI to tell the client where to find the next page in the results.

To see at how this service is implemented, continue with the next steps.

9. Press the Back button in your browser to return to the GET handler definition.

Note the Source Type is Query, this is the default Source Type, and indicates that the resource can contain zero or more results. The Pagination Size is 7, which means that there will be seven items on each page of the results. Finally, the Source for the handler looks like this:

```sql
select empno "$uri", empno, ename from (
    select emp.*,
    row_number() over (order by empno) rn
    from emp
) tmp
where
  rn between :row_offset and :row_count
```

In this query:

- The first line states that you want to return three columns. The first column is the employee id: `empno`, but aliased to a column name of `$uri` (to be explained later), the second column is again the employee ID, and the third column is the employee name, `ename`.

- Columns in result sets whose first character is $ (dollar sign) are given special treatment. They are assumed to denote columns that must be transformed into
URLs, and these are called Hyperlink Columns. Thus, naming columns with a leading $ is a way to generate hyperlinks in resources.

When a Hyperlink Column is encountered, its value is prepended with the URI of the resource in which the column is being rendered, to produce a new URI. For example, recall that the URI of this service is https://server:port/ords/workspace/hr/employees/. If the value of empno in the first row produced by this service's query is 7369, then the value of $uri becomes: https://server:port/ords/workspace/hr/employees/7369.

- JSON does not have a URI data type, so a convention is needed to make it clear to clients that a particular value represents a URI. Oracle REST Data Services uses the JSON Reference proposal, which states that any JSON object containing a member named $ref, and whose value is a string, is a URI. Thus, the column: $uri and its value: https://server:port/ords/workspace/hr/employees/7369 is transformed to the following JSON object:

```json
{
  "$uri":
   {
   "$ref":
    "https://server:port/ords/workspace/hr/employees/7369"
  }
}
```

- The inner query uses the row_number() analytical function to count the number of rows in the result set, and the outer WHERE clause constrains the result set to only return rows falling within the desired page of results. Oracle REST Data Services defines two implicit bind parameters, :row_offset and :row_count, that always contain the indices of the first and last rows that should be returned in a given page's results.

For example, if the current page is the first page and the pagination size is 7, then the value of :row_offset will be 1 and the value of :row_count will be 7.

To see a simpler way to do both hyperlinks and paged results, continue with the following steps.

10. Click on the GET handler of the employeesfeed/ resource template.

Note that the Source Type of this handler is Feed and Pagination Size is 25.

11. Change the pagination size to 7, and click Apply Changes.

The Source of the handler is just the following:

```sql
select empno, ename from emp
order by deptno, ename
```

As you can see, the query is much simpler than the previous example; however, if you click Test, you will see a result that is very similar to the result produced by the previous example.

- The Feed Source Type is an enhanced version of the Query Source Type that automatically assumes the first column in a result set should be turned into a hyperlink, eliminating the need to alias columns with a name starting with $.

In this example, the empno column is automatically transformed into a hyperlink by the Feed Source Type.

- This example demonstrates the ability of Oracle REST Data Services to automatically paginate result sets if a Pagination Size of greater than zero is defined, and the query does not explicitly dereference the :row_offset or :row_count bind parameters. Because both these conditions hold true for this...
example, Oracle REST Data Services enhances the query, wrapping it in clauses to count and constrain the number and offset of rows returned. Note that this ability to automatically paginate results also applies to the Query Source Type.

3.2.5 About cURL and Testing RESTful Services

Other sections show the testing of RESTful Services using a web browser. However, another useful way to test RESTful Services is using the command line tool named cURL.

This powerful tool is available for most platforms, and enables you to see and control what data is being sent to and received from a RESTful service. The following example uses cURL with the services mentioned in Section 3.2.4, "Exploring the Sample RESTful Services (Tutorial)".

```
$ curl -i https://server:port/ords/workspace/hr/employees/7369
```

This example produces a response like the following:

```
HTTP/1.1 200 OK
Server: Oracle-REST-Data-Services/2.0.6.78.05.25
ETag: "...
Content-Type: application/json
Transfer-Encoding: chunked
Date: Thu, 28 Mar 2014 16:49:34 GMT

{
  "empno":7369,
  "ename":"SMITH",
  "job":"CLERK",
  "mgr":7902,
  "hiredate":"1980-12-17T08:00:00Z",
  "sal":800,
  "deptno":20
}
```

The -i option tells cURL to display the HTTP headers returned by the server.

3.3 Automatic Enabling of Schema Objects for REST Access (AutoREST)

If Oracle REST Data Services has been installed on the system associated with a database connection, and if the connection is open in SQL Developer, you can use the AutoREST feature to conveniently enable or disable Oracle REST Data Services access for specified tables and views in the schema associated with that database connection.

Enabling REST access to a table or view allows it to be accessed through RESTful services.

AutoREST is a quick and easy way to expose database tables as REST resources. You sacrifice some flexibility and customizability to gain ease of effort. AutoRest lets you quickly expose data but (metaphorically) keeps you on a set of guide rails. For example, you cannot customize the output formats or the input formats, or do extra validation.

On the other hand, manually created resource modules require you to specify the SQL and PL/SQL to support the REST resources. Using resource modules requires more effort, but offers more flexibility; for example, you can customize what fields are included, do joins across multiple tables, and validate the incoming data using PL/SQL.
So, as an application developer you must make a choice: use the "guide rails" of AutoREST, or create a resource module to do exactly what you need. If you choose AutoREST, you can just enable a table (or set of tables) within a schema.

Note that enabling a schema is not equivalent to enabling all tables and views in the schema. It just means making Oracle REST Data Services aware that the schema exists and that it may have zero or more resources to expose to HTTP. Those resources may be AutoREST resources or resource module resources.

You can automatically enable Oracle REST Data Services queries to access individual database schema objects (tables and views) by using a convenient wizard in Oracle SQL Developer. (Note that this feature is only available for Oracle REST Data Services-enabled schemas, not for Oracle Application Express workspaces.)

To enable Oracle REST Data Services access to one or more specified tables or views, you must do the following in SQL Developer:

1. Enable the schema (the one associated with the connection) for REST access.
   
   **Schema** level: To enable Oracle REST Data Services access to selected objects (that you specify in the next step) in the schema associated with a connection, right-click its name in the Connections navigator and select **REST Services**, then **Enable REST Services**.
   
   (To drop support for Oracle REST Data Services access to objects in the schema associated with a connection, right-click its name in the Connections navigator and select **REST Services**, then **Drop REST Services**.)

2. Individually enable REST access for the desired objects.
   
   **Table or view** level: To enable Oracle REST Data Services access to a specified table or view, right-click its name in the Connections navigator and select **Enable REST Services**.

For detailed usage information, click the **Help** button in the wizard or dialog box in SQL Developer.

Section 3.3.1, "Examples: Accessing Objects Using RESTful Services" provides examples of using Oracle REST Data Services queries and other operations against tables and views after you have REST-enabled them.

Section 3.4, "Filtering in Queries" describes and provides examples of filtering in queries against REST-enabled tables and views.

### 3.3.1 Examples: Accessing Objects Using RESTful Services

You can automatically expose table and view objects as RESTful services using SQL Developer. This topic provides examples of accessing these RESTful services.

**Tip:** Although these examples illustrate the URL patterns used to access these resources, clients should avoid hard coding knowledge of the structure of these URLs; instead clients should follow the hyperlinks in the resources to navigate between resources. The structure of the URL patterns may evolve and change in future releases.

This topic provides examples of accessing objects using RESTful Services.

- Get Schema Metadata
- Get Object Metadata
3.3.1.1 Get Schema Metadata

This example retrieves a list of resources available through the specified schema alias. It shows RESTful services that are created by automatically enabling a table or view, along with RESTful Services that are created by resource modules.

This example retrieves a list of resources available through the specified schema alias.

**Pattern:** GET http://<HOST>:<PORT>/ords/<SchemaAlias>/metadata-catalog/

**Example:** GET http://localhost:8080/ords/ordstest/metadata-catalog/

**Result:**

```json
{
  "items": [
    {
      "name": "EMP",
      "links": [
        {
          "rel": "describes",
          "href": "http://localhost:8080/ords/ordstest/emp/"
        },
        {
          "rel": "canonical",
          "href": "http://localhost:8080/ords/ordstest/metadata-catalog/emp/",
          "mediaType": "application/json"
        }
      ]
    },
    {
      "name": "oracle.examples.hello",
      "links": [
        {
          "rel": "describes",
          "href": "http://localhost:8080/ords/ordstest/examples/hello/"
        },
        {
          "rel": "canonical",
          "href": "http://localhost:8080/ords/ordstest/metadata-catalog/examples/hello/",
          "mediaType": "application/json"
        }
      ]
    }
  ],
  "hasMore": false,
  "limit": 25,
  "offset": 0,
  "limit": 25,
  "offset": 0,
}
The list of resources includes:

- Resources representing tables or views that have been REST enabled.
- Resources defined by resource modules. Note that only resources having a concrete path (that is, not containing any parameters) will be shown. For example, a resource with a path of /module/some/path/ will be shown, but a resource with a path of /module/some/:parameter/ will not be shown.

Each available resource has two hyperlinks:

- The link with relation describes points to the actual resource.
- The link with relation canonical describes the resource.

### 3.3.1.2 Get Object Metadata

This example retrieves the metadata (which describes the object) of an individual object. The location of the metadata is indicated by the canonical link relation.

**Pattern:** GET


**Example:** GET http://localhost:8080/ords/ordstest/metadata-catalog/emp/

**Result:**

```json
{
    "name": "EMP",
    "primarykey": ["empno"],
    "members": [
        {"name": "empno", "type": "NUMBER"},
        {"name": "ename", "type": "VARCHAR2"},
        {"name": "job", "type": "VARCHAR2"},
        {"name": "mgr", "type": "NUMBER"},
    ]
}
```
3.3.1.3 Get Object Data

This example retrieves the data in the object. Each row in the object corresponds to a JSON object embedded within the JSON array

**Pattern:** GET http://<HOST>:<PORT>/ords/<SchemaAlias>/<ObjectAlias>/

**Example:** GET http://localhost:8080/ords/ordstest/emp/

**Result:**

```json
{
  "items": [
    {
      "empno": 7499,
      "ename": "ALLEN",
      "job": "SALESMAN",
      "mgr": 7698,
      "hiredate": "1981-02-20T00:00:00Z",
      "sal": 1600,
      "comm": 300,
      "deptno": 30,
      "links": [
        {
          "rel": "self",
          "href": "http://localhost:8080/ords/ordstest/emp/7499"
        }
      ]
    }
  ]
}
```
...{
  "empno": 7934,
  "ename": "MILLER",
  "job": "CLERK",
  "mgr": 7782,
  "hiredate": "1982-01-23T00:00:00Z",
  "sal": 1300,
  "comm": null,
  "deptno": 10,
  "links": [
    {
      "rel": "self",
      "href": "http://localhost:8080/ords/ordstest/emp/7934"
    }
  ],
  "hasMore": false,
  "limit": 25,
  "offset": 0,
  "count": 13,
  "links": [
    {
      "rel": "self",
      "href": "http://localhost:8080/ords/ordstest/emp/*"
    },
    {
      "rel": "edit",
      "href": "http://localhost:8080/ords/ordstest/emp/*"
    },
    {
      "rel": "describedby",
      "href": "http://localhost:8080/ords/ordstest/metadata-catalog/emp/"
    },
    {
      "rel": "first",
      "href": "http://localhost:8080/ords/ordstest/emp/*"
    }
  ]
}

### 3.3.1.4 Get Table Data Using Paging

This example specifies the offset and limit parameters to control paging of result data.

**Pattern:** GET


**Example:** GET http://localhost:8080/ords/ordstest/emp/?offset=10&limit=5

**Result:**

```json
{
  "items": [
  {
    "empno": 7900,
    "ename": "JAMES",
    "job": "CLERK",
    "mgr": 7698,
```
3.3.1.5 Get Table Data Using Query

This example specifies a filter clause to restrict objects returned.
**Pattern:** GET
http://<HOST>:<PORT>/ords/<SchemaAlias>/<ObjectAlias>/?q=<FilterClause>

**Example:** GET
http://localhost:8080/ords/ordstest/emp/?q={"deptno":{"$lte":20}}

**Result:**

```json
{
  "items": [
    {
      "empno": 7566,
      "ename": "JONES",
      "job": "MANAGER",
      "mgr": 7839,
      "hiredate": "1981-04-01T23:00:00Z",
      "sal": 2975,
      "comm": null,
      "deptno": 20,
      "links": [
        {
          "rel": "self",
          "href": "http://localhost:8080/ords/ordstest/emp/7566"
        }
      ]
    },
    ...
    {
      "empno": 7934,
      "ename": "MILLER",
      "job": "CLERK",
      "mgr": 7782,
      "hiredate": "1982-01-23T00:00:00Z",
      "sal": 1300,
      "comm": null,
      "deptno": 10,
      "links": [
        {
          "rel": "self",
          "href": "http://localhost:8080/ords/ordstest/emp/7934"
        }
      ]
    }
  ],
  "hasMore": false,
  "limit": 25,
  "offset": 0,
  "count": 7,
  "links": [
    {
      "rel": "self",
      "href": "http://localhost:8080/ords/ordstest/emp/?q=%7B%22deptno%22:%7B%22%24lte%22:20%7D" }
  ],
  {
    "rel": "edit",
    "href": "http://localhost:8080/ords/ordstest/emp/?q=%7B%22deptno%22:%7B%22%24lte%22:20%7D"
  }
}
```
3.3.1.6 Get Table Row Using Primary Key
This example retrieves an object by specifying its identifying key values.

**Pattern:** GET

http://<HOST>:<PORT>/ords/<SchemaAlias>/<ObjectAlias>/<KeyValues>

*Where* `<KeyValues>` *is a comma-separated list of key values (in key order).*

**Example:** GET http://localhost:8080/ords/ordstest/emp/7839

**Result:**

```json
{
  "empno": 7839,
  "ename": "KING",
  "job": "PRESIDENT",
  "mgr": null,
  "hiredate": "1981-11-17T00:00:00Z",
  "sal": 5000,
  "comm": null,
  "deptno": 10,
  "links": [
    {
      "rel": "self",
      "href": "http://localhost:8080/ords/ordstest/emp/7839"
    },
    {
      "rel": "edit",
      "href": "http://localhost:8080/ords/ordstest/emp/7839"
    },
    {
      "rel": "describedby",
      "href": "http://localhost:8080/ords/ordstest/metadata-catalog/emp/item"
    },
    {
      "rel": "collection",
      "href": "http://localhost:8080/ords/ordstest/emp/"
    }
  ]
}
```

3.3.1.7 Insert Table Row
This example inserts data into the object. The body data supplied with the request is a JSON object containing the data to be inserted.
If the object has a primary key, then there must be an insert trigger on the object that-populates the primary key fields. If the table does not have a primary key, then the ROWID of the row will be used as the item’s identifier.

If the object lacks a trigger to assign primary key values, then the PUT operation described in Section 3.3.1.8, "Update/Insert Table Row" should be used instead.


Example:
POST http://localhost:8080/ords/ordstest/emp/
Content-Type: application/json

{ "empno": 7, "ename": "JBOND", "job": "SPY", "deptno": 11 }

Result:

{  
  "empno": 7,
  "ename": "JBOND",
  "job": "SPY",
  "mgr": null,
  "hiredate": null,
  "sal": null,
  "comm": null,
  "deptno": 11,
  "links": [  
    {  
      "rel": "self",
      "href": "http://localhost:8080/ords/ordstest/emp/7"
    },
    {  
      "rel": "edit",
      "href": "http://localhost:8080/ords/ordstest/emp/7"
    },
    {  
      "rel": "describedby",
      "href": "http://localhost:8080/ords/ordstest/metadata-catalog/emp/item"
    },
    {  
      "rel": "collection",
      "href": "http://localhost:8080/ords/ordstest/emp/
    }
  ]
}

3.3.1.8 Update/Insert Table Row

This example inserts or updates (sometimes called an "upsert") data in the object. The body data supplied with the request is a JSON object containing the data to be inserted or updated.

Pattern: PUT
http://<HOST>:<PORT>/ords/<SchemaAlias>/<ObjectAlias>/<KeyValues>

Example:
PUT http://localhost:8080/ords/ordstest/emp/7
Content-Type: application/json

{ "empno": 7, "ename": "JBOND", "job": "SPY", "deptno": 11 }
Result:

```json
{
  "empno": 7,
  "ename": "JBOND",
  "job": "SPY",
  "mgr": null,
  "hiredate": null,
  "sal": null,
  "comm": null,
  "deptno": 11,
  "links": [
    {
      "rel": "self",
      "href": "http://localhost:8080/ords/ordstest/emp/7"
    },
    {
      "rel": "edit",
      "href": "http://localhost:8080/ords/ordstest/emp/7"
    },
    {
      "rel": "describedby",
      "href": "http://localhost:8080/ords/ordstest/metadata-catalog/emp/item"
    },
    {
      "rel": "collection",
      "href": "http://localhost:8080/ords/ordstest/emp/"
    }
  ]
}
```

### 3.3.1.9 Delete Using Filter

This example deletes object data specified by a filter clause.

**Pattern:** DELETE

http://<HOST>:<PORT>/ords/<SchemaAlias>/<ObjectAlias>/?q=<FilterClause>

**Example:** DELETE http://localhost:8080/ords/ordstest/emp/?q={"deptno":11}

**Result:**

```json
{
  "itemsDeleted": 1
}
```

### 3.3.1.10 Post by Batch Load

This example inserts object data using the batch load feature. The body data supplied with the request is a CSV file. The behavior of the batch operation can be controlled using the optional query parameters, which are described in Table 3–1.

**Pattern:** POST


**Parameters:**
Table 3–1 Parameters for batchload

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>batchesPerCommit</td>
<td>Sets the frequency for commits. Optional commit points can be set after a batch is sent to the database. The default is every 10 batches. 0 indicates commit deferred to the end of the load. Type: Integer.</td>
</tr>
<tr>
<td>batchRows</td>
<td>Sets the number of rows in each batch to send to the database. The default is 50 rows per batch. Type: Integer.</td>
</tr>
<tr>
<td>dateFormat</td>
<td>Sets the format mask for the date data type. This format is used when converting input data to columns of type date. Type: String.</td>
</tr>
<tr>
<td>delimiter</td>
<td>Sets the field delimiter for the fields in the file. The default is the comma (,).</td>
</tr>
<tr>
<td>enclosures</td>
<td>embeddedRightDouble</td>
</tr>
<tr>
<td>errors</td>
<td>Sets the user option used to limit the number of errors. If the number of errors exceeds the value specified for errorsMax (the service option) or by errors (the user option), then the load is terminated. To permit no errors at all, specify 0. To indicate that all errors be allowed (up to errorsMax value), specify UNLIMITED (-1).</td>
</tr>
<tr>
<td>errorsMax</td>
<td>A service option used to limit the number of errors allowed by users. It intended as an option for the service provider and not to be exposed as a user option. If the number of errors exceeds the value specified for errorsMax (the service option) or by errors (the user option), then the load is terminated. To permit no errors at all, specify 0. To indicate that all errors be allowed, specify UNLIMITED (-1).</td>
</tr>
<tr>
<td>lineEnd</td>
<td>Sets the line end (terminator). If the file contains standard line end characters (\r, \r\n or \n), then lineEnd does not need to be specified.</td>
</tr>
<tr>
<td>lineMax</td>
<td>Sets a maximum line length for identifying lines/rows in the data stream. A lineMax value will prevent reading an entire stream as a single line when the incorrect lineEnd character is being used. The default is unlimited.</td>
</tr>
<tr>
<td>locale</td>
<td>Sets the locale.</td>
</tr>
<tr>
<td>responseEncoding</td>
<td>Sets the encoding for the response stream.</td>
</tr>
<tr>
<td>responseFormat</td>
<td>Sets the format for response stream. This format determines how messages and bad data will be formatted. Valid values: RAW, SQL.</td>
</tr>
<tr>
<td>timestampFormat</td>
<td>Sets the format mask for the time stamp data type. This format is used when converting input data to columns of type time stamp.</td>
</tr>
<tr>
<td>timestampTZFormat</td>
<td>Sets the format mask for the time stamp time zone data type. This format is used when converting input data to columns of type time stamp time zone.</td>
</tr>
<tr>
<td>truncate</td>
<td>Indicates if and/or how table data rows should be deleted before the load. False (the default) does not delete table data before the load; True causes table data to be deleted with the DELETE SQL statement; Truncate causes table data to be deleted with the TRUNCATE SQL statement.</td>
</tr>
</tbody>
</table>

Example:

Content-Type: text/csv

empno, ename, job, mgr, hiredate, sal, comm, deptno
0, M, SPY MAST,,2005-05-01 11:00:01,4000,,11
### 3.4 Filtering in Queries

Filtering is the process of limiting a collection resource by using a per-request dynamic filter definition across multiple page resources, where each page contains a subset of items found in the complete collection. Filtering enables efficient traversal of large collections.

To filter in a query, include the parameter \( q=\text{FilterObject} \), where \( \text{FilterObject} \) is a JSON object that represents the custom selection and sorting to be applied to the resource. For example, assume the following resource:

https://example.com/ords/scott/emp/

The following query includes a filter that restricts the ENAME column to "JOHN":

https://example.com/ords/scott/emp/?\( q=\{ \text{"ENAME": "JOHN"} \} \)

- FilterObject Grammar
- Examples: FilterObject Specifications

#### 3.4.1 FilterObject Grammar

The FilterObject must be a JSON object that complies with the following syntax:

```
FilterObject { orderby, asof, wmembers }
```

The `orderby`, `asof`, and `wmembers` attributes are optional, and their definitions are as follows:

**orderby**

```
"$orderby": {orderByMembers}
```

**orderByMembers**

```
orderByProperty , orderByMembers
```

**orderByProperty**

```
columnName : sortingValue
```

**sortingValue**

- "ASC"
- "DESC"
- 
- "-1"
- 
- 1
- 
- 1

**asof**

```
"$asof": date
```
Filtering in Queries

```javascript
"$asof": "datechars"
"$asof": scn
"$asof": +int

wmembers
wpair
wpair, wmembers

wpair
columnProperty
complexOperatorProperty

columnProperty
columnName : string
columnName : number
columnName : date
columnName : simpleOperatorObject
columnName : complexOperatorObject
columnName : [complexValues]

columnName
"[\p{Alpha}][\p{Alpha}]](\{\[\p{Alnum}\]$\)+"$

complexOperatorProperty
complexKey : [complexValues]
complexKey : simpleOperatorObject

complexKey
"$and"
"$or"

complexValues
complexValue, complexValues

complexValue
simpleOperatorObject
complexOperatorObject
columnObject

columnObject
columnProperty

simpleOperatorObject
columnProperty

complexOperatorObject
columnProperty

simpleOperatorObject
"$eq" : string | number | date
"$ne" : string | number | date
"$lt" : number | date
"$lte" : number | date
"$gt" : number | date
"$gte" : number | date
"$instr" : string
"$ninstr" : string
"$like" : string
"$null" : null
"$notnull" : null
```
"$between" : betweenValue

betweenValue
  [null , betweenNotNull]
  [betweenNotNull , null]
  [betweenRegular , betweenRegular]

betweenNotNull
  number
  date

betweenRegular
  string
  number
  date

Data type definitions include the following:

string
  JSONString
number
  JSONNumber
date
  {"$date":"datechars"}
scn
  {"$scn": +int}

Where:
datechars is an RFC3339 date format in UTC (Z)

JSONString
  "
    ' chars '
chars
  char
  char chars
char
  any-Unicode-character except="-or\-or-control-character
  \'
  \\n  /\n  \b
  \f
  \n
\r
  \t
  \u four-hex-digits

JSONNumber
  int
  int frac
  int exp
  int frac exp
int
  digit
  digit1-9 digits
  - digit
  - digit1-9 digits
The FilterObject must be encoded according to Section 2.1 of RFC3986.

3.4.2 Examples: FilterObject Specifications

The following are examples of operators in FilterObject specifications.

**ORDER BY property ($orderby)**

Order by with literals

```
{ "$orderby": { "SALARY": "ASC","ENAME":"DESC"} }
```

Order by with numbers

```
{ "$orderby": {"SALARY": -1,"ENAME": 1} }
```

**ASOF property ($asof)**

With SCN (Implicit)

```
{ "$asof": 1273919 }
```

With SCN (Explicit)

```
{ "$asof": {"$scn": "1273919"}}
```

With Date (Implicit)

```
{ "$asof": "2014-06-30T00:00:00Z" }
```

With Date (Explicit)

```
{ "$asof": {"$date": "2014-06-30T00:00:00Z"}}
```
EQUALS operator ($eq)

(Implicit and explicit equality supported.)

Implicit (Support String and Dates too)

{  
  "SALARY": 1000
}

Explicit

{  
  "SALARY": {"$eq": 1000}
}

Strings

{  
  "ENAME": {"$eq": "SMITH"}
}

Dates

{  
  "HIREDATE": {"$date": "1981-11-17T08:00:00Z"}
}

NOT EQUALS operator ($ne)

Number

{  
  "SALARY": {"$ne": 1000}
}

String

{  
  "ENAME": {"$ne": "SMITH"}
}

Dates

{  
  "HIREDATE": {"$ne": {"$date": "1981-11-17T08:00:00Z"}}
}

LESS THAN operator ($lt)

(Supports dates and numbers only)

Numbers

{  
  "SALARY": {"$lt": 10000}
Dates

{  "SALARY": {"$lt": {"$date":"1999-12-17T08:00:00Z"}}
}

LESS THAN OR EQUALS operator ($lte)
(Supports dates and numbers only)

Numbers

{  "SALARY": {"$lte": 10000}
}

Dates

{  "HIREDATE": {"$lte": {"$date":"1999-12-17T08:00:00Z"}}
}

GREATER THAN operator ($gt)
(Supports dates and numbers only)

Numbers

{  "SALARY": {"$gt": 10000}
}

Dates

{  "SALARY": {"$gt": {"$date":"1999-12-17T08:00:00Z"}}
}

GREATER THAN OR EQUALS operator ($gte)
(Supports dates and numbers only)

Numbers

{  "SALARY": {"$gte": 10000}
}

Dates

{  "HIREDATE": {"$gte": {"$date":"1999-12-17T08:00:00Z"}}
}

In string operator ($instr)
(Supports strings only)

{  "ENAME": {"$instr":"MC"}
### NOT IN string operator ($ninstr)
(Supports strings only)

```
{ 'ENAME': { '$ninstr': 'MC' } }
```

### LIKE operator ($like)
(Supports strings. Escape character not supported to try to match expressions with _ or % characters.)

```
{ 'ENAME': { '$like': 'AX%' } }
```

### BETWEEN operator ($between)
(Supports string, dates, and numbers)

**Numbers**

```
{ 'SALARY': { '$between': [1000,2000] } }
```

**Dates**

```
{ 'SALARY': { '$between': [{ 'date': '1989-12-17T08:00:00Z' }, { 'date': '1999-12-17T08:00:00Z' }] } }
```

**Strings**

```
{ 'ENAME': { '$between': ['A','C'] } }
```

**Null Ranges ($lte equivalent)**
(Supported by numbers and dates only)

```
{ 'SALARY': { '$between': [null,2000] } }
```

**Null Ranges ($gte equivalent)**
(Supported by numbers and dates only)

```
{ 'SALARY': { '$between': [1000, null] } }
```

### NULL operator ($null)

```
"ENAME": {"$null": null}
}

### NOT NULL operator ($notnull)

{  "ENAME": {"$notnull": null}
}

### AND operator ($and)
(Supports all operators, including $and and $or)

**Column context delegation**
(Operators inside $and will use the closest context defined in the JSON tree.)

{  "SALARY": {"$and": [{"$gt": 1000}, {"$lt": 4000}]} }

**Column context override**
(Example: salary greater than 1000 and name like S%)  

{  "SALARY": {"$and": [{"$gt": 1000}, {"ENAME": {"$like": "S%"}}]} }

Implicit and in columns
...
{  "SALARY": [{"$gt": 1000}, {"$lt": 4000}]
}

**High order AND**
(All first columns and or high order operators -- $and and $ors -- defined at the first level of the JSON will be joined and an implicit AND)  
(Example: Salary greater than 1000 and name starts with S or T)

{  "SALARY": {"$gt": 1000},  
   "ENAME": {"$or": [{"$like": "S%"}, {"$like": "T%"}]} }

**Invalid expression (operators $lt and $gt lack column context)**

{  "$and": [{"$lt": 5000}, {"$gt": 1000}]
}

**Valid alternatives for the previous invalid expression**

{  "$and": [{"SALARY": {"$lt": 5000}}, {"SALARY": {"$gt": 1000}}]
}

{  "SALARY": [{"$lt": 5000}, {"$gt": 1000}]
}
3.5 Configuring Secure Access to RESTful Services

This section describes how to configure secure access to RESTful Services. RESTful APIs consist of resources, each resource having a unique URI. A set of resources can be protected by a privilege. A privilege defines the set of roles, at least one of which an authenticated user must possess to access a resource protected by a privilege.

Configuring a resource to be protected by a particular privilege requires creating a privilege mapping. A privilege mapping defines a set of patterns that identifies the resources that a privilege protects.

**Topics:**
- Authentication
- About Privileges for Accessing Resources
- About Users and Roles for Accessing Resources
- About the File-Based User Repository
- Tutorial: Protecting and Accessing Resources

3.5.1 Authentication

Users can be authenticated through first party cookie-based authentication or third party OAuth 2.0-based authentication.

**Topics:**
- First Party Cookie-Based Authentication
- Third Party OAuth 2.0-Based Authentication
3.5.1.1 First Party Cookie-Based Authentication

A first party is the author of a RESTful API. A first party application is a web application deployed on the same web origin as the RESTful API. A first party application is able to authenticate and authorize itself to the RESTful API using the same cookie session that the web application is using. The first party application has full access to the RESTful API.

3.5.1.2 Third Party OAuth 2.0-Based Authentication

A third party is any party other than the author of a RESTful API. A third party application cannot be trusted in the same way as a first party application; therefore, there must be a mediated means to selectively grant the third party application limited access to the RESTful API.

The OAuth 2.0 protocol (https://tools.ietf.org/html/rfc6749) defines flows to provide conditional and limited access to a RESTful API. In short, the third party application must first be registered with the first party, and then the first party (or an end user of the first party RESTful service) approves the third party application for limited access to the RESTful API, by issuing the third party application a short-lived access token.

Topics:
- Two-Legged and Three-Legged OAuth Flows

3.5.1.2.1 Two-Legged and Three-Legged OAuth Flows

Some flows in OAuth are defined as two-legged and others as three-legged.

Two-legged OAuth flows involve two parties: the party calling the RESTful API (the third party application), and the party providing the RESTful API. Two-legged flows are used in server to server interactions where an end user does not need to approve access to the RESTful API. In OAuth 2.0 this flow is called the client credentials flow. It is most typically used in business to business scenarios.

Three-legged OAuth flows involve three parties: the party calling the RESTful API, the party providing the RESTful API, and an end user party that owns or manages the data to which the RESTful API provides access. Three-legged flows are used in client to server interactions where an end user must approve access to the RESTful API. In OAuth 2.0 the authorization code flow and the implicit flow are three-legged flows. These flows are typically used in business to consumer scenarios.

For resources protected by three-legged flows, when an OAuth client is registering with a RESTful API, it can safely indicate the protected resources that it requires access to, and the end user has the final approval decision about whether to grant the client access. However for resources protected by two-legged flows, the owner of the RESTful API must approve which resources each client is authorized to access.

3.5.2 About Privileges for Accessing Resources

A privilege for accessing resources consists of the following data:
- Name: The unique identifier for the Privilege. This value is required.
- Label: The name of the privilege presented to an end user when the user is being asked to approve access to a privilege when using OAuth. This value is required if the privilege is used with a three-legged OAuth flow (see Two-Legged and Three-Legged OAuth Flows).
Description: A description of the purpose of the privilege. It is also presented to the end user when the user is being asked to approve access to a privilege. This value is required if the privilege is used with a three-legged OAuth flow.

Roles: A set of role names associated with the privilege. An authenticated party must have at least one of the specified roles in order to be authorised to access resources protected by the privilege. A value is required, although it may be an empty set, which indicates that a user must be authenticated but that no specific role is required to access the privilege.

For two-legged OAuth flows, the third party application (called a client in OAuth terminology) must possess at least one of the required roles.

For three-legged OAuth flows, the end user that approves the access request from the third party application must possess at least one of the required roles.

See also About Users and Roles for Accessing Resources.

3.5.3 About Users and Roles for Accessing Resources

A privilege enumerates a set of roles, and users can possess roles. but where are these Roles defined? What about the users that possess these roles? Where are they defined?

A privilege enumerates a set of roles, and users can possess roles. Oracle REST Data Services delegates the task of user management to the application server on which Oracle REST Data Services is deployed. Oracle REST Data Services is able to authenticate users defined and managed by the application server and to identify the roles and groups to which the authenticated user belongs. It is the responsibility of the party deploying Oracle REST Data Services on an application server to also configure the user repository on the application server.

Because an application server can be configured in many ways to define a user repository or integrate with an existing user repository, this document cannot describe how to configure a user repository in an application server. See the application server documentation for detailed information.

3.5.4 About the File-Based User Repository

Oracle REST Data Services provides a a simple file-based user repository mechanism. However, this user repository is only intended for the purposes of demonstration and testing (such as in Tutorial: Protecting and Accessing Resources), and is not supported for production use.

See the command-line help for the user command for more information on how to create a user in this repository:

```
java -jar ords.war help user
```

Format:

```
java -jar ords.war user <user> <roles>
```

Arguments:

- `<user>` is the user ID of the user.
- `<roles>` is the list of roles (zero or more) that the user has.
3.5.5 Tutorial: Protecting and Accessing Resources

This tutorial demonstrates creating a privilege to protect a set of resources, and accessing the protected resource with the following OAuth features:

- Client credentials
- Authorization code
- Implicit flow

It also demonstrates access the resource using first-party cookie-based authentication.

Topics:
- OAuth Flows and When to Use Each
- Assumptions for This Tutorial
- Steps for This Tutorial

3.5.5.1 OAuth Flows and When to Use Each

This topic explains when to use various OAuth flow features.

Use first party cookie-based authentication when accessing a RESTful API from a web application hosted on the same origin as the RESTful API.

Use the authorization code flow when you need to permit third party web applications to access a RESTful API and the third party application has its own web server where it can keep its client credentials secure. This is the typical situation for most web applications, and it provides the most security and best user experience, because the third party application can use refresh tokens to extend the life of a user session without having to prompt the user to reauthorize the application.

Use the implicit flow when the third party application does not have a web server where it can keep its credentials secure. This flow is useful for third party single-page-based applications. Because refresh tokens cannot be issued in the Implicit flow, the user will be prompted more frequently to authorize the application.

Native mobile or desktop applications should use the authorization code or implicit flows. They will need to display the sign in and authorization prompts in a web browser view, and capture the access token from the web browser view at the end of the authorization process.

Use the client credentials flow when you need to give a third party application direct access to a RESTful API without requiring a user to approve access to the data managed by the RESTful API. The third party application must be a server-based application that can keep its credentials secret. The client credentials flow must not be used with a native application, because the client credentials can always be discovered in the native executable.

3.5.5.2 Assumptions for This Tutorial

This tutorial assumes the following:

- Oracle REST Data Services is deployed at the following URL: https://example.com/ords/
- A database schema named ORDSTEST has been enabled for use with Oracle REST Data Services, and its RESTful APIs are exposed under: https://example.com/ords/ordstest/
The ORDSTEST schema contains a database table named EMP, which was created as follows:

```sql
create table emp (
    empno    number(4,0),
    ename    varchar2(10 byte),
    job      varchar2(9 byte),
    mgr      number(4,0),
    hiredate date,
    sal      number(7,2),
    comm     number(7,2),
    deptno   number(2,0),
    constraint pk_emp primary key (empno)
);
```

The resources to be protected are located under:
https://example.com/ords/ordstest/examples/employees/

### 3.5.5.3 Steps for This Tutorial

Follow these steps to protect and access a set of resources.

---

**Note:** This tutorial uses the Oracle REST Data Services API, which is described in Using the Oracle REST Data Services PL/SQL API.

---

1. **Enable the schema.** Connect to the ORDSTEST schema and execute the following PL/SQL statements:

   ```sql
   begin
   ords.enable_schema;
   commit;
   end;
   ```

2. **Create a resource.** Connect to the ORDSTEST schema and execute the following PL/SQL statements:

   ```sql
   begin
   ords.create_service(
       p_module_name => 'examples.employees' ,
       p_base_path  => '/examples/employees/' ,
       p_pattern => '.' ,
       p_items_per_page => 7 ,
       p_source  => 'select * from emp order by empno desc');
   commit;
   end;
   ```

   The preceding code creates the /examples/employees/ resource, which you will protect with a privilege in a later step.

   You can verify the resource by executing following cURL command:

   ```bash
   curl -i https://example.com/ords/ordstest/examples/employees/
   ```

   The result should be similar to the following (edited for readability):

   ```json
   Content-Type: application/json
   Transfer-Encoding: chunked

   {
   "items":
   ```
3. **Create a privilege.** While connected to the ORDSTEST schema, execute the following PL/SQL statements:

```sql
begin
    ords.create_role('HR Administrator');
    ords.create_privilege(
        p_name => 'example.employees',
        p_role_name => 'HR Administrator',
        p_label => 'Employee Data',
        p_description => 'Provide access to employee HR data');
    commit;
end;
```

The preceding code creates a role and a privilege, which belong to the ORDSTEST schema.

- The role name must be unique and must contain printable characters only.
- The privilege name must be unique and must conform to the syntax specified by the OAuth 2.0 specification, section 3.3 for scope names.
- Because you will want to use this privilege with the three-legged authorization code and implicit flows, you must provide a label and a description for the privilege. The label and description are presented to the end user during the approval phase of three-legged flows.
- The values should be plain text identifying the name and purpose of the privilege.

You can verify that the privilege was created correctly by querying the USER_ORDS_PRIVILEGES view.

```sql
select id,name from user_ords_privileges where name = 'example.employees';
```

The result should be similar to the following:
The ID value will vary from database to database, but the NAME value should be as shown.

4. **Associate the privilege with resources.** While connected to the ORDSTEST schema, execute the following PL/SQL statements:

    begin
    ords.create_privilege_mapping(
        p_privilege_name => 'example.employees',
        p_pattern => '/examples/employees/*');
    commit;
    end;

The preceding code associates the `example.employees` privilege with the resource pattern `/examples/employees/`.

You can verify that the privilege was created correctly by querying the `USER_ORDS_PRIVILEGE_MAPPINGS` view.

    select privilege_id, name, pattern from user_ords_privilege_mappings;

The result should be similar to the following:

<table>
<thead>
<tr>
<th>PRIVILEGE_ID</th>
<th>NAME</th>
<th>PATTERN</th>
</tr>
</thead>
<tbody>
<tr>
<td>10260</td>
<td><code>example.employees</code></td>
<td><code>/examples/employees/*</code></td>
</tr>
</tbody>
</table>

The PRIVILEGE_ID value will vary from database to database, but the NAME and PATTERN values should be as shown.

You can confirm that the `/examples/employees/` resource is now protected by the `example.employees` privilege by executing the following `cURL` command:

    curl -i https://example.com/ords/ordstest/examples/employees/

The result should be similar to the following (reformatted for readability):

```
HTTP/1.1 401 Unauthorized
Content-Type: text/html
Transfer-Encoding: chunked

<!DOCTYPE html>
<html>
...
</html>
```

You can confirm that the protected resource can be accessed through first party authentication, as follows.

   a. **Create an end user.** Create a test user with the HR Administrator role, required to access the `examples.employees` privilege using the file-based user repository. Execute the following command at a command prompt

       `java -jar ords.war user "hr_admin" "HR Administrator"`

       When prompted for the password, enter and confirm it.

   b. **Sign in as the end user.** Enter the following URL in a web browser:

       `https://example.com/ords/ordstest/examples/employees/`
On the page indicating that access is denied, click the link to sign in. Enter the credentials registered for the HR_ADMIN user, and click Sign In. Confirm that the page redirects to
https://example.com/ords/ordstest/examples/employees/ and that the JSON document is displayed.

5. **Register the OAuth client.** While connected to the ORDSTEST schema, execute the following PL/SQL statements:

```plsql
begin
  oauth.create_client(
    p_name => 'Client Credentials Example',
    p_grant_type => 'client_credentials',
    p_privilege_names => 'example.employees',
    p_support_email => 'support@example.com');
  commit;
end;
```

The preceding code registers a client named *Client Credentials Example*, to access the `examples.employees` privilege using the client credentials OAuth flow.

You can verify that the client was registered and has requested access to the `examples.employees` privilege by executing the following SQL statement:

```sql
select client_id,client_secret from user_ords_clients where name = 'Client Credentials Example';
```

The result should be similar to the following:

```
CLIENT_ID                        CLIENT_SECRET
-------------------------------- ------------------------
o_C2BVXKoN3tTB-IddQsQ..         4BJXceu6fbmTvruYNLig..
```

The `CLIENT_ID` and `CLIENT_SECRET` values represent the secret credentials for the OAuth client. These values must be noted and kept secure. You can think of them as the userid and password for the client application.

6. **Grant the OAuth client a required role.** While connected to the ORDSTEST schema, execute the following PL/SQL statements:

```plsql
begin
  oauth.grant_client_role('Client Credentials Example', 'HR Administrator');
  commit;
end;
```

The preceding code registers a client named *Client Credentials Example*, to access the `examples.employees` privilege using the client credentials OAuth flow.

You can verify that the client was granted the role by executing the following SQL statement:

```sql
select * from user_ords_client_roles where client_name = 'Client Credentials Example';
```

The result should be similar to the following:

```
CLIENT_ID CLIENT_NAME                 ROLE_ID   ROLE_NAME
---------- --------------------------- --------  ----------------------
10286 Client Credentials Example    10222   HR Administrator
```
7. **Obtain an OAuth access token using client credentials.**

   The OAuth protocol specifies the HTTP request that must be used to create an access token using the client credentials flow[rfc6749-4.4.].

   The request must be made to a well known URL, called the token endpoint. For Oracle REST Data Services the path of the token endpoint is always oauth/token, relative to the root path of the schema being accessed. The token endpoint for this example is:

   ```
   https://example.com/ords/ordstest/oauth/token
   ```

   Execute the following cURL command:

   ```
   curl -i --user clientId:clientSecret --data "grant_type=client_credentials"
   https://example.com/ords/ordstest/oauth/token
   ```

   In the preceding command, replace `clientId` with the CLIENT_ID value in USER_ORDS_CLIENTS for Client Credentials Example, and replace `clientSecret` with the CLIENT_SECRET value shown in USER_ORDS_CLIENTS for Client Credentials Example. The output should be similar to the following:

   ```
   HTTP/1.1 200 OK
   Content-Type: application/json
   {
     "access_token": "2YotnFZFEjr1zCsicMbpAA",
     "token_type": "bearer",
     "expires_in":3600
   }
   ```

   In the preceding output, the access token is of type `bearer`, and the value is specified by the `access_token` field. This value will be different for every request. The `expires_in` value indicates the number of seconds until the access token expires; in this case the token will expire in one hour (3600 seconds).

8. **Access a protected resource using the access token.** Execute the following cURL command:

   ```
   curl -i -H "Authorization: Bearer accessToken"
   https://example.com/ords/ordstest/examples/employees/
   ```

   In the preceding command, replace `accessToken` with the value of the `access_token` field shown in the preceding step. The output should be similar to the following:

   ```
   Content-Type: application/json
   Transfer-Encoding: chunked
   {
     "items": [
       {"empno":7934,"ename":"MILLER","job":"CLERK","mgr":7782,"hiredate":"1982-01-23T00:00:00Z","sal":1300,"comm":null,"deptno":10},
       ...
     ],
     "hasMore":true,
     "limit":7,
     "offset":0,
     "count":7,
     "links":
   }
   ```
9. Register the client for authorization code. While connected to the ORDSTEST schema, execute the following PL/SQL statements:

```plsql
begin
  oauth.create_client(
    p_name => 'Authorization Code Example',
    p_grant_type => 'authorization_code',
    p_owner => 'Example Inc.',
    p_description => 'Sample for demonstrating Authorization Code Flow',
    p_redirect_uri => 'http://example.org/auth/code/example/',
    p_support_email => 'support@example.org',
    p_support_uri => 'http://example.org/support',
    p_privilege_names => 'example.employees'
  );
  commit;
end;
```

The preceding code registers a client named Authorization Code Example, to access the examples.employees privilege using the authorization code OAuth flow. For an actual application, a URI must be provided to redirect back to with the authorization code, and a valid support email address must be supplied; however, this example uses fictitious data and the sample example.org web service.

You can verify that the client is now registered and has requested access to the examples.employees privilege by executing the following SQL statement:

```sql
select id, client_id, client_secret from user_ords_clients where name = 'Authorization Code Example';
```

The result should be similar to the following:

<table>
<thead>
<tr>
<th>ID</th>
<th>CLIENT_ID</th>
<th>CLIENT_SECRET</th>
</tr>
</thead>
<tbody>
<tr>
<td>10060</td>
<td>IGHso4B8grBC3Jw0Vx_YQ.. GefAswv8FJdMSB30Eg61Kw..</td>
<td></td>
</tr>
</tbody>
</table>

To grant access to the privilege, an end user must approve access. The CLIENT_ID and CLIENT_SECRET values represent the secret credentials for the OAuth client. These values must be noted and kept secure. You can think of them as the userid and password for the client application.

10. Obtain an OAuth access token using an authorization code. This major step involves several substeps. (You must have already created the HR_ADMIN end user in a previous step.)

   a. Obtain an OAuth authorization code.

      The end user must be prompted (via a web page) to sign in and approve access to the third party application. The third party application initiates this
process by directing the user to the OAuth Authorization Endpoint. For Oracle REST Data Services, the path of the authorization endpoint is always oauth/auth, relative to the root path of the schema being accessed. The token endpoint for this example is:

https://example.com/ords/ordstest/oauth/auth

The OAuth 2.0 protocol specifies that the Authorization request URI must include certain parameters in the query string:

The `response_type` parameter must have a value of `code`.

The `client_id` parameter must contain the value of the applications client identifier. This is the `client_id` value determined in a previous step.

The `state` parameter must contain a unique unguessable value. This value serves two purposes: it provides a way for the client application to uniquely identify each authorization request (and therefore associate any application specific state with the value; think of the value as the application's own session identifier); and it provides a means for the client application to protect against Cross Site Request Forgery (CSRF) attacks. The `state` value will be returned in the redirect URI at the end of the authorization process. The client must confirm that the value belongs to an authorization request initiated by the application. If the client cannot validate the state value, then it should assume that the authorization request was initiated by an attacker and ignore the redirect.

To initiate the Authorization request enter the following URL in a web browser:

https://example.com/ords/ordstest/oauth/auth?response_type=code&client_id=clientId&state=uniqueRandomValue

In the preceding URI, replace `clientId` with the value of the CLIENT_ID column that was noted previously, and replace `uniqueRandomValue` with a unique unguessable value. The client application must remember this value and verify it against the `state` parameter returned as part of the redirect at the end of the authorization flow.

If the `client_id` is recognized, then a sign in prompt is displayed. Enter the credentials of the HR_ADMIN end user, and click Sign In; and on the next page click Approve to cause a redirect to redirect URI specified when the client was registered. The redirect URI will include the authorization code in the query string portion of the URI. It will also include the same `state` parameter value that the client provided at the start of the flow. The redirect URI will look like the following:

http://example.org/auth/code/example/?code=D5doeTSIDgbxWiWkPi9UpA..&state=uniqueRandomValue

The client application must verify the value of the `state` parameter and then note the value of the `code` parameter, which will be used in to obtain an access token.

b. Obtain an OAuth access token.

After the third party application has an authorization code, it must exchange it for an access token. The third party application’s server must make a HTTPS request to the Token Endpoint. You can mimic the server making this request by using a cURL command as in the following example:
curl --user clientId:clientSecret --data "grant_type=authorization_code&code=authorizationCode" https://example.com/ords/ordstest/oauth/token

In the preceding command, replace clientId with the value of the CLIENT_ID shown in USER_ORDS_CLIENTS for Authorization Code Example, replace clientSecret with the value of the CLIENT_SECRET shown in USER_ORDS_CLIENTS for Authorization Code Example, and replace authorizationCode with the value of the authorization code noted in a previous step (the value of the code parameter).

The result should be similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "access_token": "psIGSSEXSBQyib0hozNEdwy.",
  "token_type": "bearer",
  "expires_in":3600,
  "refresh_token": "aRMg7AdWPuDvmieHucfV3g.."
}
```

In the preceding result, the access token is specified by the access_token field, and a refresh token is specified by the refresh_token field. This refresh token value can be used to extend the user session without requiring the user to reauthorize the third party application.

**c. Access a protected resource using the access token.**

After the third party application has obtained an OAuth access token, it can use that access token to access the protected /examples/employees/ resource:

```
curl -i -H"Authorization: Bearer accessToken" https://example.com/ords/ordstest/examples/employees/
```

In the preceding command, accessToken with the value of the access_token field shown in a previous step.

The result should be similar to the following:

```
Content-Type: application/json
Transfer-Encoding: chunked

{
  "items": [
    {
      "empno":7934,"ename":"MILLER","job":"CLERK","mgr":7782,"hiredate":"1982-01-23T00:00:00Z","sal":1300,"comm":null,"deptno":10},
      ...
    ],
  "hasMore":true,
  "limit":7,
  "offset":0,
  "count":7,
  "links": [
    {
      "rel":"self","href":"https://example.com/ords/ordstest/examples/employees/"},
    {
      "rel":"describedby","href":"https://example.com/ords/ordstest/metadata-cat"}
  ]
}
```
d. **Extend the session using a refresh token.**

At any time, the third party application can use the refresh token value to generate a new access token with a new lifetime. This enables the third party application to extend the user session at will. To do this, the third party application's server must make an HTTPS request to the Token Endpoint. You can mimic the server making this request by using a cURL command as in the following example:

```bash
curl --user clientId:clientSecret --data "grant_type=authorization_code&code=authorizationCode" https://example.com/ords/ordstest/oauth/token
```

In the preceding command, replace `clientId` with the value of the CLIENT_ID shown in USER_ORDS_CLIENTS for Client Credentials Client, replace `clientSecret` with the value of the CLIENT_SECRET shown in USER_ORDS_CLIENTS for Client Credentials Client, and replace `authorizationCode` with the value of the authorization code noted in a previous step (the value of the code parameter).

The result should be similar to the following:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "access_token": "psIGSSEXSBQyib0hozNEdw..",
  "token_type": "bearer",
  "refresh_token": "aRMg7AdWPuDvnieHucfV3g..",
  "expires_in": 3600
}
```

In the preceding result, the access token is specified by the `access_token` field, a new refresh token is specified by the `refresh_token` field. This refresh token value can be used to extend the user session without requiring the user to reauthorize the third party application. (Note that the previous access token and refresh token are now invalid; the new values must be used instead.)

11. **Register the client for implicit flow.** While connected to the ORDSTEST schema, execute the following PL/SQL statements:

```sql
begin
  oauth.create_client(
    p_name => 'Implicit Example',
    p_grant_type => 'implicit',
    p_owner => 'Example Inc.',
    p_description => 'Sample for demonstrating Implicit Flow',
    p_redirect_uri => 'http://example.org/implicit/example/',
    p_support_email => 'support@example.org',
    p_support_uri => 'http://example.org/support',
    p_privilege_names => 'example.employees'
  );
end;
```
The preceding code registers a client named Implicit Example to access the examples.employees privilege using the implicit OAuth flow. For an actual application, a URI must be provided to redirect back to with the authorization code, and a valid support email address must be supplied; however, this example uses fictitious data and the sample example.org web service.

You can verify that the client is now registered and has requested access to the examples.employees privilege by executing the following SQL statement:

```sql
SELECT id, client_id, client_secret FROM user_ords_clients WHERE name = 'Implicit Example';
```

The result should be similar to the following:

```
ID CLIENT_ID CLIENT_SECRET
---------- -------------------------------- --------------------------------
10062 7Qz--bNJPpPv8qsfNqps1A..   
```

To grant access to the privilege, an end user must approve access.

12. **Obtain an OAuth access token using implicit flow.** (You must have already created the HR_ADMIN end user in a previous step.)

   The end user must be prompted (via a web page) to sign in and approve access to the third party application. The third party application initiates this process by directing the user to the OAuth Authorization Endpoint. For Oracle REST Data Services, the path of the authorization endpoint is always oauth/auth, relative to the root path of the schema being accessed. The token endpoint for this example is:

   ```
   https://example.com/ords/ordstest/oauth/auth
   ```

   The OAuth 2.0 protocol specifies that the Authorization request URI must include certain parameters in the query string:

   - The **response_type** parameter must have a value of **token**.
   - The **client_id** parameter must contain the value of the applications client identifier. This is the **client_id** value determined in a previous step.
   - The **state** parameter must contain a unique unguessable value. This value serves two purposes: it provides a way for the client application to uniquely identify each authorization request (and therefore associate any application specific state with the value; think of the value as the application's own session identifier); and it provides a means for the client application to protect against Cross Site Request Forgery (CSRF) attacks. The **state** value will be returned in the redirect URI at the end of the authorization process. The client must confirm that the value belongs to an authorization request initiated by the application. If the client cannot validate the state value, then it should assume that the authorization request was initiated by an attacker and ignore the redirect.

   To initiate the Authorization request enter the following URL in a web browser:

   ```
   https://example.com/ords/ordstest/oauth/auth?response_type=token&client_id=clientId&state=uniqueRandomValue
   ```

   In the preceding URI, replace clientId with the value of the CLIENT_ID column that was noted previously, and replace uniqueRandomValue with a unique unguessable value. The client application must remember this value and verify it
against the `state` parameter returned as part of the redirect at the end of the authorization flow.

If the `client_id` is recognized, then a sign in prompt is displayed. Enter the credentials of the HR_ADMIN end user, and click Sign In; and on the next page click Approve to cause a redirect to redirect URI specified when the client was registered. The redirect URI will include the access token in the query string portion of the URI. It will also include the same `state` parameter value that the client provided at the start of the flow. The redirect URI will look like the following:

```http
http://example.org/auth/code/example/#access_token=D5doeTSIDgboxWkP19UpA..&type=bearer&expires_in=3600&state=uniqueRandomValue
```

The client application must verify the value of the `state` parameter and then note the value of the access token.

13. **Access a protected resource using an access token.** Execute the following `cURL` command:

```bash
curl -i -H "Authorization: Bearer accessToken"
https://example.com/ords/ordstest/examples/employees/
```

In the preceding command, replace `accessToken` with the value of the `access_token` field shown in the preceding step. The output should be similar to the following:

```
Content-Type: application/json
Transfer-Encoding: chunked

{
   "items":
       [
        {
           "empno":7934,"ename":"MILLER","job":"CLERK","mgr":7782,"hiredate":"1982-01-23T00:00:00Z","sal":1300,"comm":null,"deptno":10},
          ...
        ],
   "hasMore":true,
   "limit":7,
   "offset":0,
   "count":7,
   "links":
       [
        {
            "rel":"self","href":"https://example.com/ords/ordstest/examples/employees/"},
        {
            "rel":"describedby","href":"https://example.com/ords/ordstest/metadata-catalog/examples/employees/"},
        {
            "rel":"first","href":"https://example.com/ords/ordstest/examples/employees/"},
        {
            "rel":"next","href":"https://example.com/ords/ordstest/examples/employees/?offset=7"
        }
       ]
}
```
3.6 About Oracle REST Data Services User Roles

Oracle REST Data Services defines a small number of predefined user roles:

- **RESTful Services** - This is the default role associated with a protected RESTful service.
- **OAuth2 Client Developer** - Users who want to register OAuth 2.0 applications must have this role.
- **SQL Developer** - Users who want to use Oracle SQL Developer to develop RESTful services must have this role.
- **SODA Developer** - This is the default role that is required to access the SODA REST API. For more information about this role, see *Oracle REST Data Services SODA for REST Developer’s Guide*.
- **Listener Administrator** - Users who want to administrate an Oracle REST Data Services instance through Oracle SQL Developer must have this role. Typically, only users created through the `java -jar ords.war` user command will have this role.

Because the **Listener Administrator** role enables a user to configure an Oracle REST Data Services instance, and therefore has the capability to affect all Application Express workspaces served through that instance, Application Express users are not permitted to acquire the **Listener Administrator** role.

### 3.6.1 About Oracle Application Express Users and Oracle REST Data Services Roles

By default, Oracle Application Express users do not have any of the Oracle REST Data Services predefined user roles. This means that, by default, Application Express users cannot:

- Invoke protected RESTful Services
- Register OAuth 2.0 applications
- Use Oracle SQL Developer to develop RESTful services.

This applies to all Application Express users, including Application Express developers and administrators. It is therefore important to remember to follow the steps below to add Application Express users to the appropriate user groups, so that they can successfully perform the above actions.

### 3.6.1.1 Granting Application Express Users Oracle REST Data Services Roles

To give an Application Express User any of the roles above, the user must be added to the equivalent Application Express user group. For example, to give the **RESTEASY_ADMIN** user the **RESTful Services** role, follow these steps:

1. Log in to the **RESTEASY** workspace as a **RESTEASY_ADMIN**.
2. Navigate to Administration and then Manage Users and Groups.
3. Click the Edit icon to the left of the RESTEASY_ADMIN user.
4. For User Groups, select RESTful Services.
5. Click Apply Changes.

### 3.6.1.2 Automatically Granting Application Express Users Oracle REST Data Services Roles

Adding Application Express users to the appropriate user groups can be an easily overlooked step, or can become a repetitive task if there are many users to be managed.

To address these issues, you can configure Oracle REST Data Services to automatically grant Application Express users a predefined set of RESTful Service roles by modifying the defaults.xml configuration file.

In that file, Oracle REST Data Services defines three property settings to configure roles:

- **apex.security.user.roles** - A comma separated list of roles to grant ordinary users, that is, users who are not developers or administrators.
- **apex.security.developer.roles** - A comma separated list of roles to grant users who have the Developer account privilege. Developers also inherit any roles defined by the `apex.security.user.roles` setting.
- **apex.security.administrator.roles** - A comma separated list of roles to grant users who have the Administrator account privilege. Administrators also inherit any roles defined by the `apex.security.user.roles` and `apex.security.developer.roles` settings.

For example, to automatically give all users the RESTful Services privilege and all developers and administrators the OAuth2 Client Developer and SQL Developer roles, add the following to the defaults.xml configuration file:

```xml
<!-- Grant all Application Express Users the ability to invoke protected RESTful Services -->
<entry key="apex.security.user.roles">RESTful Services</entry>
<!-- Grant Application Express Developers and Administrators the ability to register OAuth 2.0 applications and use Oracle SQL Developer to define RESTful Services -->
<entry key="apex.security.developer.roles">OAuth2 Client Developer, SQL Developer</entry>
```

Oracle REST Data Services must be restarted after you make any changes to the defaults.xml configuration file.

### 3.6.2 Controlling RESTful Service Access with Roles

The built-in RESTful Service role is a useful default for identifying users permitted to access protected RESTful services.

However, it will often also be necessary to define finer-grained roles to limit the set of users who may access a specific RESTful service.

**Topics:**

- About Defining RESTful Service Roles
- Associating Roles with RESTful Privileges
3.6.2.1 About Defining RESTful Service Roles
A RESTful Service role is an Application Express user group. To create a user group to control access to the Gallery RESTful Service, follow these steps. (The step details here use the image gallery application in Appendix C, "Development Tutorial: Creating an Image Gallery" as an example.)

1. Log in to the RESTEASY workspace as a workspace administrator.
2. Navigate to Administration and then Manage Users and Groups.
3. Click the Groups tab.
4. Click Create User Group.
5. For Name, enter Gallery Users.
6. Click Create Group.

3.6.2.2 Associating Roles with RESTful Privileges
After a user group has been created, it can be associated with a RESTful privilege. To associate the Gallery Users role with the example.gallery privilege, follow these steps. (The step details here use the image gallery application in Appendix C, "Development Tutorial: Creating an Image Gallery" as an example.)

1. Navigate to SQL Workshop and then RESTful Services.
2. In the Tasks section, click RESTful Service Privileges.
3. Click Gallery Access.
4. For Assigned Groups, select Gallery Users.
5. Click Apply Changes.

With these changes, users must have the Gallery Users role to be able to access the Gallery RESTful service.

3.7 Authenticating Against WebLogic Server and GlassFish User Repositories
Oracle REST Data Services can use APIs provided by WebLogic Server and GlassFish to verify credentials (username and password) and to retrieve the set of groups and roles that the user is a member of.

This section walks through creating a user in the built-in user repositories provided by WebLogic Server and GlassFish, and verifying the ability to authenticate against that user.

This document does not describe how to integrate WebLogic Server and GlassFish with the many popular user repository systems such as LDAP repositories, but Oracle REST Data Services can authenticate against such repositories after WebLogic Server or GlassFish has been correctly configured. See your application server documentation for more information on what user repositories are supported by the application server and how to configure access to these repositories.

Topics:
- Authenticating Against WebLogic Server
- Authenticating Against GlassFish
3.7.1 Authenticating Against WebLogic Server

Authenticating a user against WebLogic Server involves the following major steps:

1. **Creating a WebLogic Server User**
2. **Verifying the WebLogic Server User**

3.7.1.1 Creating a WebLogic Server User

To create a sample WebLogic Server user, follow these steps:

1. Start WebLogic Server if it is not already running
2. Access the WebLogic Server Administration Console (typically http://server:7001/console), enter your credentials.
3. In the navigation tree on the left, click the **Security Realms** node
4. If a security realm already exists, go to the next step. If a security realm does not exist, create one as follows:
   a. Click **New**.
   b. For **Name**, enter Test-Realm, then click **OK**.
   c. Click **Test-Realm**.
   d. Click the **Providers** tab.
   e. Click **New**, and enter the following information:
      - **Name**: test-authenticator
      - **Type**: DefaultAuthenticator
   f. Restart WebLogic Server if you are warned that a restart is necessary.
   g. Click **Test-Realm**.
5. Click the **Users and Groups** tab.
6. Click **New**, and enter the following information:
   - **Name**: 3rdparty_dev2
   - **Password**: Enter and confirm the desired password for this user.
7. Click **OK**.
8. Click the **Groups** tab.
9. Click **New**, and enter the following information:
   - **Name**: OAuth2 Client Developer (case sensitive)
10. Click **OK**.
11. Click the **Users** tab.
12. Click the 3rdparty_dev2 user.
13. Click the **Groups** tab.
14. In the **Chosen** list, add OAuth2 Client Developer.
15. Click **Save**.

You have created a user named 3rdparty_dev2 and made it a member of a group named OAuth2 Client Developer. This means the user will acquire the OAuth2
Client Developer role, and therefore will be authorized to register OAuth 2.0 applications.

Now verify that the user can be successfully authenticated, as explained in Section 3.7.1.2, "Verifying the WebLogic Server User".

### 3.7.1.2 Verifying the WebLogic Server User

To verify that the WebLogic Server user created in Section 3.7.1.1, "Creating a WebLogic Server User" can be successfully authenticated, follow these steps:

1. In your browser, go to a URI in the following format:
   
   `https://server:port/ords/resteasy/ui/oauth2/clients/`

2. Enter the credentials of the `3rdparty_dev2` user, and click Sign In.

The OAuth 2.0 Client Registration page should be displayed, with no applications listed. If this page is displayed, you have verified that authentication against the WebLogic Server user repository is working.

However, if the sign-on prompt is displayed again with the message User is not authorized to access resource, then you made mistake (probably misspelling the Group List value).

### 3.7.2 Authenticating Against GlassFish

Authenticating a user against GlassFish involves the following major steps:

1. Creating a GlassFish User
2. Verifying the GlassFish User

#### 3.7.2.1 Creating a GlassFish User

To create a sample GlassFish user, follow these steps:

1. Start GlassFish if it is not already running
2. Access the GlassFish Administration Console (typically `http://server:4848`); and if you have configured a password, enter your credentials.
3. Navigate to the Security Configuration pages:
4. In the navigation tree on the left, expand the Configurations node, and then expand the following nodes: server-config, Security, Realms, file.
5. Click Manage Users.
6. Click New, and enter the following information:
   - **Name**: `3rdparty_dev2`
   - **Group List**: OAuth2 Client Developer (case sensitive)
   - **Password**: Enter and confirm the desired password for this user.
7. Click OK.

You have created a user named `3rdparty_dev2` and made it a member of a group named OAuth2 Client Developer. This means the user will acquire the OAuth2 Client Developer role, and therefore will be authorized to register OAuth 2.0 applications.

Now verify that the user can be successfully authenticated, as explained in Section 3.7.2.2, "Verifying the GlassFish User".
3.7.2.2 Verifying the GlassFish User
To verify that the WebLogic Server user created in Section 3.7.2.1, "Creating a GlassFish User" can be successfully authenticated, follow these steps:

1. In your browser, go to a URI in the following format:
   https://server:port/ords/resteasy/ui/oauth2/clients/
2. Enter the credentials of the 3rdparty_dev2 user, and click Sign In.

   The OAuth 2.0 Client Registration page should be displayed, with no applications listed. If this page is displayed, you have verified that authentication against the WebLogic Server user repository is working.

   However, if the sign-on prompt is displayed again with the message User is not authorized to access resource, then you made mistake (probably misspelling the Group List value).

3.8 Integrating with Existing Group/Role Models
The examples in other sections demonstrate configuring the built-in user repositories of WebLogic Server and GlassFish. In these situations you have full control over how user groups are named. If a user is a member of a group with the exact same (case sensitive) name as a role, then the user is considered to have that role.

However, when integrating with existing user repositories, RESTful service developers will often not have any control over the naming and organization of user groups in the user repository. In these situations a mechanism is needed to map from existing "physical" user groups defined in the user repository to the "logical" roles defined by Oracle REST Data Services and/or RESTful Services.

In Oracle REST Data Services, this group to role mapping is performed by configuring a configuration file named role-mapping.xml.

Topics:
- About role-mapping.xml

3.8.1 About role-mapping.xml
role-mapping.xml is a Java XML Properties file where each property key defines a pattern that matches against a set of user groups, and each property value identifies the roles that the matched user group should be mapped to. It must be located in the same folder as the defaults.xml configuration file. The file must be manually created and edited.

Consider this example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
  <entry key="webdevs">RESTful Services</entry>
</properties>
```

This role mapping is straightforward, stating that any user who is a member of a group named: webdevs is given the role RESTful Services, meaning that all members of the webdevs group can invoke RESTful Services.

A mapping can apply more than one role to a group. For example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
```
This rule gives members of the webdevs group both the RESTful Services and SQL Developer roles.

Topics:
- Parameterizing Mapping Rules
- Dereferencing Parameters
- Indirect Mappings

3.8.1.1 Parameterizing Mapping Rules

Having to explicitly map from each group to each role may not be scalable if the number of groups or roles is large. To address this concern, you can parameterize rules. Consider this example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
  <entry key="{prefix}.webdevs">RESTful Services</entry>
</properties>
```

This example says that any group name that ends with .webdevs will be mapped to the RESTful Services role. For example, a group named: HQ.webdevs would match this rule, as would a group named: EAST.webdevs.

The syntax for specifying parameters in rules is the same as that used for URI Templates; the parameter name is delimited by curly braces ({}).

3.8.1.2 Dereferencing Parameters

Any parameter defined in the group rule can also be dereferenced in the role rule. Consider this example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
  <entry key="cn={userid},ou={group},dc=MyDomain,dc=com">{group}</entry>
</properties>
```

This example maps the organizational unit component of an LDAP distinguished name to a role. It says that the organizational unit name maps directly to a role with same name. Note that it refers to a {userid} parameter but never actually uses it; in effect, it uses {userid} as a wildcard flag.

For example, the distinguished name cn=jsmith,ou=Developers,dc=MyDomain,dc=com will be mapped to the logical role named Developers.

3.8.1.3 Indirect Mappings

To accomplish the desired role mapping, it may sometimes be necessary to apply multiple intermediate rules. Consider this example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
```

This rule can be used to map a group name that contains a wildcard to a role name that also contains a wildcard.
This example maps the organizational unit component of an LDAP distinguished name to some roles. Complicating matters is the fact that users can come from two different organizations, resulting in differing distinguishing name patterns.

- Users from example.com always have a single common name (CN) identifying their user id, followed by the organizational unit (OU) and the domain name (DC). For example: `cn=jsmith,ou=Developers,dc=example,dc=com`.
- Users from acquired.com have varying numbers of common name (CN) prefixes, but the organizational unit is the field you are interested in. For example: `cn=ProductDev,cn=abell,ou=Engineering,dc=acquired,dc=com`.
- Both organizations identify software engineers with `ou=Developers`.

You want to map engineers in both organizations to the `RESTful Services` and `SQL Developer` roles.

- The first rule maps engineers in the example.com organization to the intermediate `Developers` role.
- The second rule maps engineers in the acquired.com organization to the intermediate `Developers` role.
- The final rule maps from the intermediate `Developers` role to the `RESTful Services` and `SQL Developer` roles.

### 3.9 Using the Oracle REST Data Services PL/SQL API

Oracle REST Data Services has a PL/SQL API (application programming interface) that you can use as an alternative to the SQL Developer graphical interface for many operations. The available subprograms are included in the following PL/SQL packages:

- ORDS, documented in Chapter 5, "ORDS PL/SQL Package Reference"
- OAUTH, documented in Chapter 6, "OAUTH PL/SQL Package Reference"

To use the Oracle REST Data Services PL/SQL API, you must first:

- Install Oracle REST Data Services in the database that you will use to develop RESTful services.
- Enable one or more database schemas for REST access (see Section 3.3, "Automatic Enabling of Schema Objects for REST Access (AutoREST)").

**Topics:**

- Creating a RESTful Service Using the PL/SQL API
- Testing the RESTful Service

#### 3.9.1 Creating a RESTful Service Using the PL/SQL API

You can create a RESTful service by connecting to a REST-enabled schema and using the `ORDS.CREATE_SERVICE` procedure.

The following example creates a simple "Hello-World"-type service:

```sql
begin
```
The preceding example does the following:

- Creates a resource module named examples.routes.
- Sets the base path (also known as the URI prefix) of the module to /examples/routes/.
- Creates a resource template in the module, with the route pattern greeting/:name.
- Creates a GET handler and sets its source as a SQL query that forms a short greeting:
  - GET is the default value for the p_method parameter, and it is used here because that parameter was omitted in this example.
  - COLLECTION_FEED is the default value for the p_method parameter, and it is used here because that parameter was omitted in this example.
- An optional parameter named whom is specified.

3.9.2 Testing the RESTful Service

To test the RESTful service that you created in Section 3.9.1, "Creating a RESTful Service Using the PL/SQL API", start Oracle REST Data Services if it is not already started:

```
java -jar ords.war
```

Enter the URI of the service in a browser. The following example displays a "Hello" greeting to Joe, by default from the current user because no whom parameter is specified:

```
http://localhost:8080/ords/ordstest/examples/routes/greeting/Joe
```

In this example:

- Oracle REST Data Services is running on localhost and listening on port 8080.
- Oracle REST Data Services is deployed at the context-path /ords.
- The RESTful service was created by a database schema named ordstest.
- Because the URL does not include the optional whom parameter, the :whom bind parameter is bound to the null value, which causes the query to use the value of the current database user (sys_context('USERENV','CURRENT_USER')) instead.

If you have a JSON viewing extension installed in your browser, you will see a result like the following:

```
{
  "items": [
  {
    "greeting": "Hello Joe from ORDSTEST"
  }
```


The next example is like the preceding one, except the optional parameter `whom` is specified to indicate that the greeting is from Jane.

```
```

This time, the result will look like the following:

```
{
  "items": [
    {
      "greeting": "Hello Joe from Jane"
    }
  ],
  "hasMore": false,
  "limit": 25,
  "offset": 0,
  "count": 1,
  "links": [
    {
      "rel": "self",
      "href": "http://localhost:8080/ords/ordstest/examples/routes/greeting/"
    },
    {
      "rel": "describedby",
      "href": "http://localhost:8080/ords/ordstest/metadata-catalog/examples/routes/greeting/"
    },
    {
      "rel": "first",
      "href": "http://localhost:8080/ords/ordstest/examples/routes/greeting/Joe"
    }
  ]
}
```

Notice that in this result, what follows "from" is Jane and not ORDSTEST, because the `:whom` bind parameter was bound to the Jane value.
This chapter explains how to use Oracle REST Data Services to provide HTTP-based REST access to Oracle NoSQL Database (also referred to here as NoSQL) data and metadata.

See also some NoSQL example files included in the Oracle REST Data Services installation. For a short document about how to get started accessing NoSQL stores, double-click the file examples/getting-started-nosql/index.html under the location where you installed Oracle REST Data Services.

**Topics:**
- About NoSQL and Oracle REST Data Services
- NoSQL Store Installation and Registration
- Adding and Removing a NoSQL Store for Use with Oracle REST Data Services
- NoSQL REST Services

### 4.1 About NoSQL and Oracle REST Data Services

Oracle REST Data Services can provide HTTP-based REST service access to Oracle NoSQL Database, through operations on the NoSQL data and metadata. Data access consists of CRUD (create, read, update, delete) operations on NoSQL tables:

- **Create:** HTTP POST and PUT to provide services to create row data.
- **Read:** HTTP GET to provide key, partial key, and field range queries on row data, with directives for consistency and order.
- **Update:** HTTP PUT to provide services to update row data.
- **Delete:** HTTP DELETE to provide services to delete row data.

Metadata access consists of HTTP GET to provide services to read NoSQL metadata.

**Topics:**
- Technology Environment
- Typical Oracle REST Data Services Input and Output

### 4.1.1 Technology Environment

The key pieces in the technology environment for using Oracle REST Data Services with NoSQL are:

- REST services
Direct web service calls to Oracle NoSQL Database, and marshall data returned into JSON format.

Can be deployed using Oracle WebLogic Server, GlassFish, or Apache Tomcat.

- Oracle NoSQL Database (Release 3.2.5 or later)
  Provides full CRUD (create, read, update, delete) operations across single and master/replicated nodes.
  Provides access through the Oracle NoSQL Database driver (.jar file).

The essential components of this environment are:

- A client, such as Java, JavaScript, or C++
- A web application server, such as standalone (Jetty), Oracle WebLogic Server, GlassFish, Apache Tomcat
- Oracle REST Data Services: the HTTP-based REST service provider
- NoSQL servlets: Oracle REST Data Services components to provide CRUD operations on the NoSQL database
- Oracle NoSQL Database: distributed master/replica NoSQL database that is the target of the NoSQL servlets

Figure 4–1 provides an overview of a typical environment for Oracle REST Data Services with NoSQL.

**Figure 4–1  Overview of Typical Environment**
This figure shows a typical environment in which Oracle REST Data Services is used with Oracle NoSQL Database. The figure is explained in text before and after the graphic.

In Figure 4–1:
- Oracle REST Data Services is inside the web server.
- Clients submit HTTP requests to Oracle REST Data Services inside the web server, and receive responses. In this case the HTTP requests are targeting NoSQL stores DB1 and DB2.
- Oracle REST Data Services users and roles are used by Oracle REST Data Services.
- Two NoSQL stores (named DB1 and DB2) communicate with Oracle REST Data Services inside the web server.
- Configuration information for the two NoSQL stores is used by Oracle REST Data Services. The DB1 store is secure (includes the NoSQL user, roles, and SSL details), and the DB2 store is non-secure. The configuration information was specified the stores were registered with Oracle REST Data Services.

4.1.2 Typical Oracle REST Data Services Input and Output

Example 4–1 is a REST service request (input) to get metadata about tables from a NoSQL database.

Example 4–1  Getting Metadata from a NoSQL Database
http://localhost:8080/ords/sales/metadata-catalog/

In Example 4–1:
- http://localhost:8080/ords is the URL for accessing the Oracle REST Data Services web application from the web server running on localhost with port 8080.
- /sales is the NoSQL database identifier. This identifier will map to a NoSQL configuration.
- metadata-catalog/ is the component that services the metadata request.

Example 4–1 might product output similar to the following in response:

```json
{
  "items": [
    {
      "name": "complexUsers",
      "links": [
        {
          "rel": "canonical",
          "href": "http://localhost:8080/ords/sales/metadata-catalog/tables/complexUsers/",
          "mediaType": "application/json"
        },
        {
          "rel": "describes",
          "href": "http://localhost:8080/ords/sales/tables/complexUsers/"
        }
      ]
    }
  ]
}
```
4.2 NoSQL Store Installation and Registration

This topic supplements the information in Chapter 1, "Installing Oracle REST Data Services".

To install and register a NoSQL store involves regarding the Oracle REST Data Services installation host as a NoSQL Client. Therefore, if the NoSQL store is secure, then the security properties file (client.security) and the SSL information file (client.trust) must be copied to the Oracle REST Data Services host's network. These files are generated when the NoSQL store is first established, as explained in the NoSQL documentation:


If the NoSQL store is secure, it must be accessed by a NoSQL user. NoSQL provide a Wallet (for Enterprise Editions) and a password file (for Community editions) to hold password information. The password file is not secure because it is plain text, so Oracle REST Data Services provides an alternative storage of passwords. Creating the Wallet is also described in the NoSQL documentation; and if the Wallet is used, it too must be copied into the Oracle REST Data Services host's network.

Each Oracle REST Data Services installation has a configuration directory, which includes a nosql directory for NoSQL store configurations. The nosql directory will have one directory for each NoSQL store, and each store-specific directory includes the following files:

- client.security (for Secure noSQL stores)
- client.trust (for Secure NoSQL stores)
- Wallet directory (Optional for NoSQL secure stores)
- nosql.properties

The nosql.properties file includes the following:

- oracle.dbtools.kv.store.name=<storename>
- oracle.dbtools.kv.store.hosts=<comma separated set of HOSTS:PORTS>
  [And optionally the following:]
- oracle.dbtools.kv.store.pagelimit=<maximum number of items per page>
- oracle.dbtools.kv.store.security.properties=client.security  (the name of the NoSQL client.security file)
- oracle.dbtools.kv.store.requiredRoles=<list of comma separated roles>
If `oracle.dbtools.kv.store.requiredRoles` property is not included in the `nosql.properties` file, then the REST service is authorized for anonymous users; otherwise access is granted to any user that has any of these roles.

The `client.security` file contains the SSL properties, and it also contains the following:

```plaintext
oracle.kv.auth.username=<NoSQL user used for ORDS to access the store>
```

The `client.security` file may also need to include the following:

- `oracle.kv.auth.wallet.dir=<wallet directory>` to specify the wallet directory, if a wallet is used
- `oracle.dbtools.kv.store.auth.pwd=<encrypted pwd>` to specify the password for the NoSQL user, if a password option is used

Note all path names in the `client.security` file must be specified with the full path.

The following examples show:

- A store accessed by Oracle REST Data Services anonymous users on the localhost port 5000 with name `kvstore` (a typical KVLite store).

```
ords/nosql/sales
  nosql.properties
    oracle.dbtools.kv.store.name=kvstore
    oracle.dbtools.kv.store.hosts=localhost:5000
```

- A store accessed by Oracle REST Data Services users with the role `SALES` on the localhost port 5000 with name `kvstore` (a typical KVLite store).

```
ords/nosql/sales
  nosql.properties
    oracle.dbtools.kv.store.name=kvstore
    oracle.dbtools.kv.store.hosts=localhost:5000
    oracle.dbtools.kv.store.requiredRoles=SALES
```

- A secure store on `xyz6160769` port 5000 accessed with user `NOSQL_PUBLIC_USER` and whose password is stored in a wallet. It can be accessed by anonymous Oracle REST Data Services users.

```
ords/nosql/sales
  nosql.properties
    oracle.dbtools.kv.store.name=mystore
    oracle.dbtools.kv.store.hosts=xyz6160769.example.com:5000
    oracle.dbtools.kv.store.security.properties=client.security

  client.security
    oracle.kv.ssl.trustStore=d:/nosqlconfs/ords/nosql/salesecure/client.trust
    oracle.kv.transport=ssl
    oracle.kv.ssl.protocols=TLSv1.2,TLSv1.1,TLSv1
    oracle.kv.ssl.hostnameVerifier=dnmatch(CN=NoSQL)
    oracle.kv.auth.username=NOSQL_PUBLIC_USER
    oracle.kv.auth.wallet.dir=d:/nosqlconfs/ords/nosql/salesecure/wallet
    wallet (directory)
```

- A secure store on `xyz6160769` port 5000 accessed with user `NOSQL_PUBLIC_USER` and whose password is stored in the properties file (encoded) It can be accessed by anonymous Oracle REST Data Services users.

```
ords/nosql/sales
  nosql.properties
```

---

NoSQL and Oracle REST Data Services 4-5
4.3 Adding and Removing a NoSQL Store for Use with Oracle REST Data Services

To add a NoSQL store, use the `java` command with the `nosqladd` keyword.

Usage:
```
<hostPorts> <roles>
```

Options:
- `[--secure]` Flag indicating that the store is secure.
- `[--clientTrustFile]` Location of the client.trust file, which will be copied from this location into the NoSQL configuration area.
- `[--clientSecurityFile]` Location of the client.security file, which will be copied from this location, with potential updates, to the NoSQL configuration area.
- `[--walletDir]` Location of the wallet directory (folder), which will be copied from this location to the NoSQL configuration area.
- `[--password]` Password for the connection.
- `[--user]` User name for the connection.
- `[--pagelimit]` Maximum number of items per page for this store.

Arguments:
- `<storeAlias>` Store alias.
- `<storeName>` Store name.
- `<hostPorts>` Host:Port values (comma-separated list).
- `<roles>` Roles (comma-separated list).

To remove (delete) a NoSQL store, use the `java` command with the `nosqldel` keyword.

Usage:
```
java -jar ords.war nosqldel <storeAlias>
```

Arguments:
- `<storeAlias>` Store alias.

4.4 NoSQL REST Services

A NoSQL request has the following format:
host:port/ords/<nosqlalias>/<nosqlservice>

Where:
- <nosqlalias> is the alias for the NoSQL store.
- <nosqlservice> is an appropriate string for the metadata, data, or DDL service being invoked.

Topics:
- Metadata Services
- Data Services
- DDL Services
- Common Parameters

4.4.1 Metadata Services

GET (Read Metadata) is the only NoSQL service available.

Topics:
- Read Metadata (GET)

4.4.1.1 Read Metadata (GET)

The GET (Read Metadata) service has different <nosqlservice> formats and output, depending on whether the request is for metadata for all tables or a specified table.

- For all tables: <nosqlservice> = metadata-catalog/

The JSON output is of the following format, which has links to each table’s metadata and the first page of data. For example:

```json
{
   "items": [
      
      { 
         "name": "complexUsers",
         "links": [
            { "rel": "canonical",
              "href": "http://localhost:8080/ords/sales/metadata-catalog/tables/complexUsers/",
              "mediaType": "application/json"
            },
            { "rel": "describes",
              "href": "http://localhost:8080/ords/sales/tables/complexUsers/"
            }
         ]
      },
      
      { 
         "name": "shardUsers",
         "links": [
            { "rel": "canonical",
              "href": "http://localhost:8080/ords/sales/metadata-catalog/tables/shardUsers/",
              "mediaType": "application/json"
            }
         ]
      }
   ]
}```
For a specified table: `<nosqlservice> = metadata-catalog/tables/<name>/`
where `<name>` is the name of the table

The JSON output is of the following format, with properties, keys, column details, child tables, and links. For example:

```json
{
  "type": "TABLE",
  "name": "simpleUsers",
  "fullname": "simpleUsers",
  "description": null,
  "primarykey": [
    "userID"
  ],
  "shardkey": [
    "userID"
  ],
  "members": [
    {
      "name": "firstName",
      "type": "STRING",
      "description": null
    }
  ]
}
```
For a specified table's items:  

\[ \text{<nosqlservice>} = \text{metadata-catalog/tables/}<\text{name}>/item \text{ where } <\text{name}> \text{ is the name of the table} \]

The JSON output is of the following format, with properties, keys, column details, child tables, and links. For example:

\[
\{
    "name": "simpleUsers",
    "fullname": "simpleUsers",
    "description": null,
    "primarykey": [
        "userID"
    ],
    "indexes": [
        {
            "name": "compoundIndex",
            "description": null,
            "indexfields": [  
                "lastName",
                "firstName"
            ]
        },
        {
            "name": "simpleIndex",
            "description": null,
            "indexfields": [  
                "firstName"
            ]
        }
    ],
    "childtables": [],
    "links": [
        {
            "rel": "collection",
            "href": "http://localhost:8080/ords/sales/metadata-catalog/"
        },
        {
            "rel": "canonical",
            "href": "http://localhost:8080/ords/sales/metadata-catalog/tables/simpleUsers/",
            "mediaType": "application/json"
        },
        {
            "rel": "describes",
            "href": "http://localhost:8080/ords/sales/tables/simpleUsers/"
        }
    ]
\}
\]
4.4.2 Data Services

Services are available for reading, writing, and deleting NoSQL data.

Topics:
- Read Data (GET)
- Idempotent Write (PUT)
- Write (POST)
- Delete (DELETE)

4.4.2.1 Read Data (GET)

The GET (Read Data) service has different <nosqlservice> formats and output, depending on whether the request is for all data in a table or a subset of the data in a table.

- For all data in a table: <nosqlservice> = tables/<name>/

The JSON output is of the following format. The data is returned in an unordered sequence, and it includes property pair values and links from the collection to show where it belongs and that it is editable. For example:

```
{
  "items": [
    {
      "$version":
```
"firstName": "Bob",
"lastName": "Johnson",
"userID": 109,
"links": [
  {
    "href": "http://localhost:8080/ords/sales/tables/simpleUsers/109",
    "rel": 'self'
  },
  {
    "href": "http://localhost:8080/ords/sales/metanadata-catalog/tables/simpleUsers/item",
    "rel": 'describedby'
  }
],

"firstName": "Alex",
"lastName": "Robertson",
"userID": 1,
"links": [
  {
    "href": "http://localhost:8080/ords/sales/tables/simpleUsers/1",
    "rel": 'self'
  },
  {
    "href": "http://localhost:8080/ords/sales/metadata-catalog/tables/simpleUsers/item",
    "rel": 'describedby'
  }
]
For data in a table where keys are specified: `<nosqlservice> = tables/<name>/</keys>`

`<name>` is the name of the table, and `<keys>` is a set of comma-separated index key values. The key values must be in the order of key definition. If not all key parts are specified, retrieval is based on partial keys. If the full key is specified, the output is a single item.

The JSON output includes property pair values and links from the collection to show where it belongs. The following example shows JSON output when a full key is specified.

```
{
  "$version": "r00ABXcsAAQC5a2Pp3hHLajI2Ew7k4e1AAAAMMIB4wAAAAEAAABAAAAAAE7Ng.",
  "firstName": "Alex",
  "lastName": "Robertson",
  "userID": 1,
  "accepts": [
    "application/json"
  ],
  "links": [
    {
      "href": "http://localhost:8080/ords/sales/tables/simpleUsers/1",
      "rel": "self"
    },
    {
      "href": "http://localhost:8080/ords/sales/metadata-catalog/tables/simpleUsers/item",
      "rel": "describedby"
    },
    {
      "href": "http://localhost:8080/ords/sales/tables/simpleUsers/1",
      "rel": "edit",
      "targetSchema": "application/json"
    },
    {
      "href": "http://localhost:8080/ords/sales/tables/simpleUsers/",
      "rel": "collection"
    }
  ]
}
```

The following example shows JSON output where a partial key is specified.

```
{
  "items": [
    {
      "$version": "r00ABXcsAAQC5a2Pp3hHLajI2Ew7k4e1AAAAMMIB4wAAAAEAAABAAAAAAE7Lw.",
      "firstName": "Alex",
      "lastName": "Robertson",
      "email": "alero@email.com",
      "links": [
        {
          "href": "http://localhost:8080/ords/sales/tables/shardUsers/Robertson,Alex",
          "rel": "self"
        },
        {
          "href": "http://localhost:8080/ords/sales/metadata-catalog/tables/shardUsers/item",
          "rel": "describedby"
        }
      ]
    }
  ]
}
```
For data in a table where an index and keys are specified: <nosqlservice> = tables/<name>/index/<indexname>/<keys>

:name is the name of the table, <indexname> is the name of the index, and <keys> is a set of comma-separated index key values. The key values must be in the order of key definition. If not all key parts are specified, retrieval is based on partial keys. If the full key is specified, the output is a single item.
The JSON output includes property pair values and links from the collection to show where it belongs and that it is editable. For example:

```json
{
  "items": [
    {
      "$version": "r00ABXcsAAQC5a2Fp3hHLaJ1Z2Etw7k4e1AAAAAAAALIBAwAAAAAABAAAABAAAAAEaIk.",
      "firstName": "Joel",
      "lastName": "Robertson",
      "userID": 3,
      "links": [
        {
          "href": "http://localhost:8080/ords/sales/tables/simpleUsers/3",
          "rel": "self"
        },
        {
          "href": "http://localhost:8080/ords/sales/metadata-catalog/tables/simpleUsers/item",
          "rel": "describedby"
        }
      ]
    },
    {
      "$version": "r00ABXcsAAQC5a2Fp3hHLaJ1Z2Etw7k4e1AAAAAAAALIBAwAAAAAABAAAABAAAAAEa4k.",
      "firstName": "Joel",
      "lastName": "Jones",
      "userID": 6,
      "links": [
        {
          "href": "http://localhost:8080/ords/sales/tables/simpleUsers/6",
          "rel": "self"
        },
        {
          "href": "http://localhost:8080/ords/sales/metadata-catalog/tables/simpleUsers/item",
          "rel": "describedby"
        }
      ]
    }
  ],
  "hasMore": false,
  "count": 2,
  "limit": 25,
  "offset": 0,
  "links": [
    {
      "href": "http://localhost:8080/ords/sales/metadata-catalog/tables/simpleUsers/",
      "rel": "describedby"
    },
    {
      "href": "http://localhost:8080/ords/sales/tables/simpleUsers/index/simpleIndex/Joel",
      "rel": "self"
    }
  ]
}
```
4.4.2  Idempotent Write (PUT)

The PUT (Idempotent Write) service has different formats and output, depending on whether the operation is upsert, put if present, or put if version. ("Put if version" means that the PUT operation will succeed only if the latest version in the NoSQL database is the stated version. This prevents a client from overwriting a newer value with an older value.)

- For Upset (update it present, create if not): <nosqlservice> = tables/<name>
  The body of the request is JSON format of the form returned by the Read Data (GET) operations, but without links. For example:
  
  ```json
  { "userID": 108, "firstName": "Bob", "lastName": "Johnson" }
  ```

- For Put if Present: <nosqlservice> = tables/<name>/<key>
  <name> is the name of the table, and <key> is the primary key value of the new object.
  The body of the request is JSON format of the form returned by the Read Data (GET) operations, but without links. For example:
  
  ```json
  { "userID": 108, "firstName": "Bob", "lastName": "Johnson" }
  ```
  If the row is not present, the status 404 (NOT_FOUND) is returned.

- For Put if Present for a specified version: <nosqlservice> = tables/<name>/<key>
  <name> is the name of the table, <key> is the primary key value of the new object, and the header parameter If-Match specifies the value of the $version property on an item. If the version matches the one specified, then it is updated.
  The body of the request is JSON format of the form returned by the Read Data (GET) operations, but without links. For example:
  
  ```json
  { "userID": 108, "firstName": "Bob", "lastName": "Johnson" }
  If-Match= r00ABXcsAAQc5a2Fp3hLajI2Ew7k4e1AAAAAAAL4AwAAAABAAAAAABa4k
  ```
  If the row is not present, the status 404 (NOT_FOUND) is returned. If the row does not match the specified version, the status 412 (PRECONDITION_FAILED) is returned.

4.4.3  Write (POST)

The POST (Put if Absent) service has the following format:

<nosqlservice> = tables/<name>/

<name> is the name of the table.

The body of the request is JSON format of the form returned by the Read Data (GET) operations, but without links. For example:

```json
{ "userID": 108, "firstName": "Bob", "lastName": "Johnson" }
```

If the row is present, the status 400 ((BAD_REQUEST) This row already exists) is returned. If the row is absent, the status 201 (CREATED) is returned.
4.4.2.4 Delete (DELETE)
The DELETE (Delete) service can have an optional version identifier specified.

- For Delete with no version specified: `<nosqlservice> = tables/<name>/<keys>
  <name>` is the name of the table, and `<keys>` is the primary key value or set of comma-separated primary key values of the rows to be deleted. If not all key parts are specified, deletion is based on partial keys.

  If a specified row is present, the status 204 (NO_CONTENT) is returned. If a specified row is absent, the status 404 (NOT_FOUND) is returned.

- For Delete if the version and keys must result in only one object being returned (that is, the object is fully specified): `<nosqlservice> = tables/<name>/<keys`
  `<name>` is the name of the table, `<keys>` is the primary key value or set of comma-separated primary key values of the rows to be deleted, and the header parameter `If-Match` specifies the value of the `$version` property on an item. If the version matches the one specified, then it is deleted. For example:

  `If-Match= r00ABXcsAAQC5a2Pp3bHLajJZEs7k4elAAAAAAAAA4BAwAAAAEAAAABAAAAAAAEAk`

  If a specified row-version combination is present, the status 204 (NO_CONTENT) is returned. If a specified row is present but not for the specified version, the status 412 (PRECONDITION_FAILED) is returned. If a specified row is absent, the status 404 (NOT_FOUND) is returned.

4.4.3 DDL Services
Services are available for manipulating NoSQL tables and indexes.

For table requests, the body of the request is a text format corresponding to that specified in Getting Started with Oracle NoSQL Database Tables (http://docs.oracle.com/cd/NOSQL/html/GettingStartedGuideTables/index.html), Appendix A (“Table Data Definition Language Overview”).

Topics:
- Create/Update (POST)
- Drop (POST)
- Poll (GET)

4.4.3.1 Create/Update (POST)
For DDL create and update operations, the service has the following `<nosqlservice>` format: `<nosqlservice> = ddl/

Example:

http://localhost:8080/ords/sales/ddl/

The body of the request contains the DDL to be executed on the NoSQL store. Examples of the body of the request:

```
CREATE TABLE IF NOT EXISTS myUsers (firstName STRING,lastName STRING,userID INTEGER,PRIMARY KEY (userID))

CREATE INDEX IF NOT EXISTS myUser_index ON myUsers(lastName, firstName)

ALTER TABLE myUsers(ADD newUser-column STRING)
```
If the operation succeeds, a status of OK is returned, along with a Location header variable giving the location of the new resource. For example (creating an index):

Status: 200: OK
Headers:
Location: http://localhost:8080/ords/sales/metadata-catalog/tables/myUsers/index/myUser_index

If the operation fails, an error code is returned.

If the operation does not complete within 1 second, a status of ACCEPTED is returned, along with a task ID so that you can send requests to see if it has finished. For example (creating an index):

Status: 202: Accepted
Headers:
Location: http://localhost:8080/ords/sales/ddl/tasks/16

4.4.3.2 Drop (POST)
For DDL drop operations, the service has the following <nosqlservice> format:
<nosqlservice> = ddl/

Example:
http://localhost:8080/ords/sales/ddl/

The body of the request contains the DDL to be executed on the NoSQL store.
Examples of the body of the request:
DROP INDEX IF EXISTS myUser_index ON myUsers
DROP TABLE IF EXISTS myUsers

If the operation succeeds, a status of NO_CONTENT is returned.
If the operation fails, an error code is returned.
If the operation does not complete within 1 second, a status of ACCEPTED is returned, along with a task ID so that you can send requests to see if it has finished. For example (dropping a table):

Status: 202: Accepted
Headers:
Location: http://localhost:8080/ords/sales/ddl/tasks/17

4.4.3.3 Poll (GET)
The Poll service has the following <nosqlservice> format: <nosqlservice> = ddl/tasks/<id>

Example:
http://localhost:8080/ords/sales/ddl/tasks/17

If the operation succeeds, a status of OK is returned, and the Location header can provide additional information.
If the Location header is the same as the URL request (for example, http://localhost:8080/ords/sales/ddl/tasks/17), then the task is still ongoing.
If the Location header is blank or null, then the task has finished and this task was a drop operation, so there is no new location.
If the `Location` header is not blank and is not the same as the URL request, then it is the location of the new or updated resource.

### 4.4.4 Common Parameters

Several groups of parameters can be used for with multiple operations and for specific desired results.

**Topics:**
- Parameters to Skip Results and Limit the Number of Results
- Parameters to Specify Query Conditions
- Parameters to Specify Direction (Order), and Consistency and Durability Guarantees

#### 4.4.4.1 Parameters to Skip Results and Limit the Number of Results

In GET operations, you can specify a limit and an offset by using `?limit={limit}&offset={offset}`. For example:

```text
<nosqlservice> = metadata-catalog/?limit=2&offset=1
```

The preceding example returns two objects per page, starting with the second object (that is, skipping the first one). The JSON output might be as follows:

```json
{
  "items": [
    {
      "name": "evolveUsers",
      "links": [
        {
          "rel": "canonical",
          "href": "http://localhost:8080/ords/sales/metadata-catalog/tables/evolveUsers/",
          "mediaType": "application/json"
        },
        {
          "rel": "describes",
          "href": "http://localhost:8080/ords/sales/tables/evolveUsers/"
        }
      ]
    },
    {
      "name": "shardUsers",
      "links": [
        {
          "rel": "canonical",
          "href": "http://localhost:8080/ords/sales/metadata-catalog/tables/shardUsers/",
          "mediaType": "application/json"
        },
        {
          "rel": "describes",
          "href": "http://localhost:8080/ords/sales/tables/shardUsers/"
        }
      ],
      "hasMore": false,
      "count": 2,
    }
  ]
}
```
4.4.4.2 Parameters to Specify Query Conditions
You can limit rows retrieved by using field ranges, and you can specify the order of retrieval. The query structure for this is a simple list of ANDed pairs. For example:

?q={"firstName":{"$between":["A","C"],"$direction":"reverse"}

Special characters must be escaped using URL encoding. For example:

http://localhost:8080/ords/sales/tables/simpleUsers/index/simpleIndex/?q={"firstName":{"$between":%5B"A","C"%5D}}

4.4.4.3 Parameters to Specify Direction (Order), and Consistency and Durability Guarantees
You can specify values for the direction (order), consistency guarantees, and durability guarantees, overriding any related default values in the store.

- For direction (order), specify the $direction property with one of the values as follows:

  $direction = "forward" | "reverse" | "unordered"

For direction (order), the default is "forward".

For primary key retrieval using a partial or empty key, only "unordered" is allowed, and any other value is ignored.

- For consistency guarantees, specify the $consistency property with appropriate values as follows (note that <$LONG> means a long number):

  $consistency = <Absolute> | <None> | <Replica> | <Time> | <Version>
  <Absolute> = "absolute"
  <None> = "none"
  <Replica> = "replica"
  <Time> = <Lag>","<Timeout>
  <Lag> = <$LONG>","<Units>
  <Timeout> = <$LONG>","<Units>
  <Units> = "nanoseconds" | "microseconds" | "milliseconds" | "seconds" | "minutes" | "hours" | "days"
  <version> = <version_id>','$<VERSION>
  <version_id> = version id of item as returned in GET

For example:

http://localhost:8080/ords/sales/tables/shardUsers/Robertson,Beatrix?q={"$consi
For durability guarantees, specify a header called `NoSQL-Write-Option` with a value in the following format (note that `<LONG>` means a long number):

\[
\text{<WriteOption> = (\langle\text{DurabilityPolicy}\rangle | \langle\text{DurabilityGuarantees}\rangle) ;}
\]

\[
\begin{align*}
\langle\text{WriteTimeOut}\rangle &; \\
\langle\text{DurabilityPolicy}\rangle & = \langle\text{Durability}\rangle = \langle\text{DurabilityValue}\rangle \\
\langle\text{DurabilityValue}\rangle & = \langle\text{COMMIT\_NO\_SYNC}\rangle | \langle\text{COMMIT\_SYNC}\rangle | \langle\text{COMMIT\_WRITE\_NO\_SYNC}\rangle \\
\langle\text{DurabilityGuarantees}\rangle & = \langle\text{MasterSyncPolicy}\rangle ; \langle\text{ReplicaSyncPolicy}\rangle ; \langle\text{ReplicaAckPolicy}\rangle \\
\langle\text{MasterSyncPolicy}\rangle & = \langle\text{MasterSync}\rangle = \langle\text{NO\_SYNC}\rangle | \langle\text{SYNC}\rangle | \langle\text{WRITE\_NO\_SYNC}\rangle \\
\langle\text{ReplicaSyncPolicy}\rangle & = \langle\text{ReplicaSync}\rangle = \langle\text{NO\_SYNC}\rangle | \langle\text{SYNC}\rangle | \langle\text{WRITE\_NO\_SYNC}\rangle \\
\langle\text{ReplicaAckPolicy}\rangle & = \langle\text{ReplicaAck}\rangle = \langle\text{ALL}\rangle | \langle\text{NONE}\rangle | \langle\text{SIMPLE\_MAJORITY}\rangle \\
\langle\text{WriteTimeOut}\rangle & = \langle\text{TimeOut}\rangle = \langle\text{LONG}\rangle [ ; \langle\text{Units}\rangle = \langle\text{LONG}\rangle ]
\end{align*}
\]

For example, create a header `NoSQL-Write-Option` with the value:

\[
\text{Durability = COMMIT\_NO\_SYNC; TimeOut = 3; Units = seconds}
\]
The ORDS PL/SQL package contains subprograms (procedures and functions) for developing RESTful services using Oracle REST Data Services.

The Oracle REST Data Services PL/SQL API is explained in Section 3.9, "Using the Oracle REST Data Services PL/SQL API".
ORDS.CREATE_ROLE

Format

ORDS.CREATE_ROLE(
    p_role_name  VARCHAR2 (255) IN);

Description

CREATE_ROLE creates an Oracle REST Data Services role with the specified name.

Parameters

    p_role_name
Name of the role.

Usage Notes

After the role is created, it can be associated with any Oracle REST Data Services privilege.
Note: ORDS.CREATE_SERVICE is deprecated. Use ORDS.DEFINE_SERVICE instead.

Format

```sql
ORDS.CREATE_SERVICE(
    p_module_name VARCHAR2 IN,
    p_base_path   VARCHAR2 IN,
    p_pattern     VARCHAR2 IN,
    p_method      VARCHAR2 IN DEFAULT 'GET',
    p_source_type VARCHAR2 IN DEFAULT ords.source_type_collection_feed,
    p_source      VARCHAR2 IN ords_handlers.source%type ,
    p_items_per_page INTEGER IN DEFAULT 25,
    p_status          VARCHAR2 IN DEFAULT 'PUBLISHED',
    p_etag_type      VARCHAR2 IN DEFAULT 'HASH',
    p_etag_query     VARCHAR2 IN DEFAULT NULL ,
    p_mimes_allowed  VARCHAR2 IN DEFAULT NULL ,
    p_module_comments VARCHAR2 IN DEFAULT NULL,
    p_template_comments VARCHAR2 IN DEFAULT NULL,
    p_handler_comments  VARCHAR2 IN DEFAULT NULL );
```

Description

Creates a new RESTful service.

Examples

The following example creates a simple service.

```sql
EXECUTE ORDS.CREATE_SERVICE(
    p_module_name => 'examples.routes' ,
    p_base_path   => '/examples/routes/',
    p_pattern     => 'greeting/:name',
    p_source => 'select 'Hello ' || :name || '' from '' || nvl(:whom,sys_context('USERENV','CURRENT_USER')) "greeting" from dual');
```
ORDS.DEFINE_HANDLER

Format

```sql
ORDS.DEFINE_HANDLER(
    p_module_name VARCHAR2(255) IN,
    p_pattern     VARCHAR2(600) IN,
    p_method      VARCHAR2(10) IN DEFAULT 'GET',
    p_source_type VARCHAR2(30) IN DEFAULT ords.source_type_collection_feed,
    p_source      CLOB IN ords_handlers.source%type,
    p_items_per_page  INTEGER IN DEFAULT 25,
    p_mimes_allowed VARCHAR2(255) IN DEFAULT NULL,
    p_comments  VARCHAR2(4000) IN DEFAULT NULL);
```

Description

DEFINE_HANDLER defines a module handler. If the handler already exists, then the handler and any existing handlers will be replaced by this definition; otherwise, a new handler is created.

Parameters

**p_module_name**
Name of the owning RESTful service module. Case sensitive.

**p_pattern**
Matching pattern for the owning resource template.

**p_method**
The HTTP Method to which this handler will respond. Valid values: GET (retrieves a representation of a resource), POST (creates a new resource or adds a resource to a collection), PUT (updates an existing resource), DELETE (deletes an existing resource).

**p_source_type**
The HTTP request method for this handler. Valid values:

- **source_type_collection_feed**: Executes a SQL query and transforms the result set into a ORDS Standard JSON representation. Available when the HTTP method is GET. Result Format: JSON
- **source_type_collection_item**: Executes a SQL query returning one row of data into a ORDS Standard JSON representation. Available when the HTTP method is GET. Result Format: JSON
- **source_type_media**: Executes a SQL query conforming to a specific format and turns the result set into a binary representation with an accompanying HTTP Content-Type header identifying the Internet media type of the representation. Result Format: Binary
- **source_type_plsql**: Executes an anonymous PL/SQL block and transforms any OUT or IN/OUT parameters into a JSON representation. Available only when the HTTP method is DELETE, PUT, or POST. Result Format: JSON
- **source_type_query || source_type_csv_query**: Executes a SQL query and transforms the result set into either an ORDS legacy JavaScript Object Notation (JSON) or CSV representation, depending on the format selected. Available when the HTTP method is GET. Result Format: JSON or CSV
- `source_type_query_one_row`. Executes a SQL query returning one row of data into an ORDS legacy JSON representation. Available when the HTTP method is GET. Result Format: JSON

- `source_type_feed`. Executes a SQL query and transforms the results into a JSON Feed representation. Each item in the feed contains a summary of a resource and a hyperlink to a full representation of the resource. The first column in each row in the result set must be a unique identifier for the row and is used to form a hyperlink of the form: `path/to/feed/{id}`, with the value of the first column being used as the value for `{id}`. The other columns in the row are assumed to summarize the resource and are included in the feed. A separate resource template for the full representation of the resource should also be defined. Result Format: JSON

**p_source**
The source implementation for the selected HTTP method.

**p_items_per_page**
The default pagination for a resource handler HTTP operation GET method, that is, the number of rows to return on each page of a JSON format result set based on a database query. Default: NULL (defers to the resource module setting).

**p_mimes_allowed**
Comma-separated list of MIME types that the handler will accept. Applies to PUT and POST only.

**p_comments**
Comment text.

**Usage Notes**
Only one handler for each HTTP method (source type) is permitted.
ORDS.DEFINE_MODULE

Format

```sql
ORDS.DEFINE_MODULE(
    p_module_name    VARCHAR2(255) IN,
    p_base_path      VARCHAR2(255) IN,
    p_items_per_page NUMBER(0) IN DEFAULT 25,
    p_status         VARCHAR2(30) IN DEFAULT 'PUBLISHED',
    p_comments       VARCHAR2(4000) IN DEFAULT NULL );
```

Description

DEFINE_MODULE defines a resource module. If the module already exists, then the module and any existing templates will be replaced by this definition; otherwise, a new module is created.

Parameters

- **p_module_name**
  Name of the owning RESTful service module. Case sensitive.

- **p_base_path**
  The base of the URI that is used to access this RESTful service. Example: `hr/` means that all URIs starting with `hr/` will be serviced by this resource module.

- **p_items_per_page**
  The default pagination for a resource handler HTTP operation GET method, that is, the number of rows to return on each page of a JSON format result set based on a database query. Default: NULL (defers to the resource module setting).

- **p_status**
  Publication status. Valid values: `PUBLISHED` (default) or `NOT_PUBLISHED`.

- **p_comments**
  Comment text.
ORDS.DEFINE_PARAMETER

Format

ORDS.DEFINE_PARAMETER(
    p_module_name         VARCHAR2(255) IN,
    p_pattern             VARCHAR2(600) IN,
    p_method              VARCHAR2(10) IN DEFAULT 'GET',
    p_name                VARCHAR2(100) IN,
    p_bind_variable_name  VARCHAR2(30) IN DEFAULT NULL,
    p_source_type         VARCHAR2(20) IN DEFAULT 'HEADER',
    p_param_type          VARCHAR2(30) IN DEFAULT 'STRING',
    p_access_method       VARCHAR2(10) IN DEFAULT 'IN',
    p_comments            VARCHAR2(4000) IN DEFAULT NULL );

Description

DEFINE_PARAMETER defines a module handler parameter. If the parameter already exists, then the parameter will be replaced by this definition; otherwise, a new parameter is created.

Parameters

p_module_name
Name of the owning RESTful service module. Case sensitive.

p_pattern
Matching pattern for the owning resource template.

p_method
The owning handler HTTP Method. Valid values: GET (retrieves a representation of a resource), POST (creates a new resource or adds a resource to a collection), PUT (updates an existing resource), DELETE (deletes an existing resource).

p_name
The name of the parameter, as it is named in the URI Template or HTTP Header. Used to map names that are not valid SQL parameter names.

p_bind_variable_name
The name of the parameter, as it will be referred to in the SQL. If NULL is specified, then the parameter is unbound.

p_source_type
The type that is identified if the parameter originates in the URI Template or a HTTP Header. Valid values: HEADER, RESPONSE, URI.

p_param_type
The native type of the parameter. Valid values: STRING, INT, DOUBLE, BOOLEAN, LONG, TIMESTAMP.

p_access_method
The parameter access method. Indicates if the parameter is an input value, output value, or both. Valid values: IN, OUT, INOUT.

p_comments
Comment text.
ORDS.DEFINE_PRIVILEGE

Format

ORDS.DEFINE_PRIVILEGE(
    p_privilege_name  VARCHAR2 (255) IN,
    p_roles           PL/SQL TABLE (VC_ARR) IN,
    p_patterns        PL/SQL TABLE (VC_ARR) IN,
    p_modules         PL/SQL TABLE (VC_ARR) IN,
    p_label           VARCHAR2 (255)        IN     DEFAULT NULL,
    p_description     VARCHAR2 (4000)       IN     DEFAULT NULL,
    p_comments        VARCHAR2 (4000)       IN     DEFAULT NYLL);

OR

ORDS.DEFINE_PRIVILEGE(
    p_privilege_name  VARCHAR2 (255) IN,
    p_roles           PL/SQL TABLE (VC_ARR) IN,
    p_patterns        PL/SQL TABLE (VC_ARR) IN,
    p_label           VARCHAR2 (255)        IN     DEFAULT NULL,
    p_description     VARCHAR2 (4000)       IN     DEFAULT NULL,
    p_comments        VARCHAR2 (4000)       IN     DEFAULT NYLL);

OR

ORDS.DEFINE_PRIVILEGE(
    p_privilege_name  VARCHAR2 (255) IN,
    p_roles           PL/SQL TABLE (VC_ARR) IN,
    p_patterns        PL/SQL TABLE (VC_ARR) IN,
    p_label           VARCHAR2 (255)        IN     DEFAULT NULL,
    p_description     VARCHAR2 (4000)       IN     DEFAULT NULL,
    p_comments        VARCHAR2 (4000)       IN     DEFAULT NYLL);

Description

DEFINE_PRIVILEGE defines an Oracle REST Data Services privilege. If the privilege already exists, then the privilege and any existing patterns and any associations with modules and roles will be replaced by this definition; otherwise, a new privilege is created.

Parameters

p_privilege_name
Name of the privilege. No spaces allowed.

p_roles
The names of the roles, at least one of which the privilege requires. May be empty, in which case the user must be authenticated but does not require any specific role; however, must not be null. Unauthenticated users will be denied access.

p_patterns
A list of patterns.

p_modules
A list of module names referencing modules created for the current schema.

p_label
Name of this security constraint as displayed to an end user. May be null.

p_description
A brief description of the purpose of the resources protected by this constraint.
p_comments
Comment text.
ORDS.DEFINE_SERVICE

Format

ORDS.DEFINE_SERVICE(
  p_module_name VARCHAR2(255) IN,
  p_base_path VARCHAR2(255) IN,
  p_pattern VARCHAR2(600) IN,
  p_method VARCHAR2(10) IN DEFAULT 'GET',
  p_source_type VARCHAR2(30) IN DEFAULT ords.source_type_collection_feed,
  p_source CLOB IN,
  p_items_per_page INTEGER IN DEFAULT 25,
  p_status VARCHAR2(30) IN DEFAULT 'PUBLISHED',
  p_etag_type VARCHAR2(30) IN DEFAULT 'HASH',
  p_etag_query VARCHAR2(4000) IN DEFAULT NULL,
  p_mimes_allowed VARCHAR2(255) IN DEFAULT NULL,
  p_module_comments VARCHAR2(4000) IN DEFAULT NULL,
  p_template_comments VARCHAR2(4000) IN DEFAULT NULL,
  p_handler_comments VARCHAR2(4000) IN DEFAULT NULL);

Description

DEFINE_SERVICE defines a resource module, template, and handler in one convenience call. If the module already exists, then the module and any existing templates will be replaced by this definition; otherwise, a new module is created.

Parameters

p_module_name
Name of the RESTful service module. Case sensitive. Must be unique.

p_base_path
The base of the URI that is used to access this RESTful service. Example: hr/ means that all URIs starting with hr/ will be serviced by this resource module.

p_pattern
A matching pattern for the resource template. For example, a pattern of /objects/:object/:id? will match /objects/emp/101 (matches a request for the item in the emp resource with id of 101) and will also match /objects/emp/. (Matches a request for the emp resource, because the :id parameter is annotated with the ? modifier, which indicates that the id parameter is optional.)

p_method
The HTTP Method to which this handler will respond. Valid values: GET (retrieves a representation of a resource), POST (creates a new resource or adds a resource to a collection), PUT (updates an existing resource), DELETE (deletes an existing resource).

p_source_type
The HTTP request method for this handler. Valid values:

- source_type_collection_feed. Executes a SQL query and transforms the result set into an ORDS Standard JSON representation. Available when the HTTP method is GET. Result Format: JSON

- source_type_collection_item. Executes a SQL query returning one row of data into a ORDS Standard JSON representation. Available when the HTTP method is GET. Result Format: JSON
■ `source_type_media`. Executes a SQL query conforming to a specific format and turns the result set into a binary representation with an accompanying HTTP Content-Type header identifying the Internet media type of the representation. Result Format: Binary

■ `source_type_plsql`. Executes an anonymous PL/SQL block and transforms any OUT or IN/OUT parameters into a JSON representation. Available only when the HTTP method is DELETE, PUT, or POST. Result Format: JSON

■ `source_type_query || source_type_csv_query`. Executes a SQL query and transforms the result set into either an ORDS legacy JavaScript Object Notation (JSON) or CSV representation, depending on the format selected. Available when the HTTP method is GET. Result Format: JSON or CSV

■ `source_type_query_one_row`. Executes a SQL query returning one row of data into an ORDS legacy JSON representation. Available when the HTTP method is GET. Result Format: JSON

■ `source_type_feed`. Executes a SQL query and transforms the results into a JSON Feed representation. Each item in the feed contains a summary of a resource and a hyperlink to a full representation of the resource. The first column in each row in the result set must be a unique identifier for the row and is used to form a hyperlink of the form: `path/to/feed/{id}`, with the value of the first column being used as the value for `{id}`. The other columns in the row are assumed to summarize the resource and are included in the feed. A separate resource template for the full representation of the resource should also be defined. Result Format: JSON

**p_source**
The source implementation for the selected HTTP method.

**p_status**
Publication status. Valid values: PUBLISHED (default) or NOT_PUBLISHED.

**p_items_per_page**
The default pagination for a resource handler HTTP operation GET method, that is, the number of rows to return on each page of a JSON format result set based on a database query. Default: NULL (defers to the resource module setting).

**p_etag_type**
A type of entity tag to be used by the resource template. An entity tag is an HTTP Header that acts as a version identifier for a resource. Use entity tag headers to avoid retrieving previously retrieved resources and to perform optimistic locking when updating resources. Valid values are HASH, QUERY, NONE:

- **HASH** (known as Secure HASH): The contents of the returned resource representation are hashed using a secure digest function to provide a unique fingerprint for a given resource version.
- **QUERY**: Manually define a query that uniquely identifies a resource version. A manually defined query can often generate an entity tag more efficiently than hashing the entire resource representation.
- **NONE**: Do not generate an entity tag.

**p_etag_query**
Query that is used to generate the entity tag.
**p_mimes_allowed**
Comma-separated list of MIME types that the handler will accept. Applies to PUT and POST only.

**p_module_comments**
Comment text.

**p_template_comments**
Comment text.

**p_handler_comments**
Comment text.

**Usage Notes**
Use this procedure instead of the deprecated ORDS.CREATE_SERVICE procedure.

**Examples**
The following example creates a simple service.

```sql
EXECUTE ORDS.DEFINE_SERVICE(
    p_module_name => 'examples.routes',
    p_base_path   => '/examples/routes/',
    p_pattern     => 'greeting/:name',
    p_source => 'select ''Hello '' || :name || '' from '' || nvl(:whom,sys_context(''USERENV'',''CURRENT_USER''))' greeting' from dual');
```
ORDS.DEFINE_TEMPLATE

Format

ORDS.DEFINE_TEMPLATE(
    p_module_name VARCHAR2(255) IN,
    p_pattern     VARCHAR2(600) IN,
    p_priority    NUMBER(0) IN DEFAULT 0,
    p_etag_type   VARCHAR2(30) IN DEFAULT 'HASH',
    p_etag_query  VARCHAR2(400) IN DEFAULT NULL,
    p_comments    VARCHAR2(4000) IN DEFAULT NULL);

Description

DEFINE_TEMPLATE defines a resource template. If the template already exists, then the template and any existing handlers will be replaced by this definition; otherwise, a new template is created.

Parameters

p_module_name
Name of the owning RESTful service module. Case sensitive.

p_pattern
A matching pattern for the resource template. For example, a pattern of /objects/:object/:id? will match /objects/emp/101 (matches a request for the item in the emp resource with id of 101) and will also match /objects/emp/ (Matches a request for the emp resource, because the :id parameter is annotated with the ? modifier, which indicates that the id parameter is optional.)

p_priority
The priority for the order of how the resource template should be evaluated: 0 (low priority, the default) through 9 (high priority).

p_etag_type
A type of entity tag to be used by the resource template. An entity tag is an HTTP Header that acts as a version identifier for a resource. Use entity tag headers to avoid retrieving previously retrieved resources and to perform optimistic locking when updating resources. Valid values are HASH, QUERY, NONE:

- HASH (known as Secure HASH): The contents of the returned resource representation are hashed using a secure digest function to provide a unique fingerprint for a given resource version.
- QUERY: Manually define a query that uniquely identifies a resource version. A manually defined query can often generate an entity tag more efficiently than hashing the entire resource representation.
- NONE: Do not generate an entity tag.

p_etag_query
Query that is used to generate the entity tag.

p_comments
Comment text.
ORDS.DELETE_MODULE

Format

ORDS.DELETE_MODULE(
    p_module_name  VARCHAR2(255)  IN);

Description

DELETE_MODULE deletes a resource module.

Parameters

p_module_name
Name of the owning RESTful service module. Case sensitive.

Usage Notes

If the module does not already exist or is accessible to the current user, then no exception is raised.
ORDS.DELETE_PRIVILEGE

Format

ORDS.DELETE_PRIVILEGE(
    p_name  VARCHAR2(255) IN);

Description

DELETE_PRIVILEGE deletes a resource module.

Parameters

p_name
Name of the privilege.

Usage Notes

If the module does not already exist or is accessible to the current user, then no exception is raised.
ORDS.DELETE_ROLE

Format

```
ORDS.DELETE_ROLE(
    p_role_name  VARCHAR2(255) IN);
```

Description

DELETE_ROLE deletes the named role.

Parameters

- **p_name**
  Name of the role.

Usage Notes

This will also delete any association between the role and any privileges that reference the role.
ORDS.DROP_REST_FOR_SCHEMA

Format

ORDS.DROP_REST_FOR_SCHEMA(
   p_schema VARCHAR2(30) IN);

Description

DROP_REST_FOR_SCHEMA deletes all auto-REST Oracle REST Data Services metadata for the associated schema.

Parameters

p_name
Name of the schema.

Usage Notes

This procedure effectively "undoes" the actions performed by the ORDS.ENABLE_SCHEMA procedure.

Examples

The following example deletes all auto-REST Oracle REST Data Services metadata for the SCOTT schema.

EXECUTE ORDS.DROP_REST_FOR_SCHEMA('scott');
ORDS.ENABLE_OBJECT

Format

ORDS.ENABLE_OBJECT(
    p_enabled         BOOLEAN  IN DEFAULT TRUE,
    p_schema          VARCHAR2(30) IN DEFAULT NULL,
    p_object          VARCHAR2(255) IN,
    p_object_type     VARCHAR2(30) IN DEFAULT 'TABLE',
    p_object_alias    VARCHAR2(255) IN DEFAULT NULL,
    p_auto_rest_auth  BOOLEAN  IN DEFAULT NULL);

Description

ENABLE_OBJECT enables Oracle REST Data Services access to a specified table or view in a schema.

Parameters

p_enabled
TRUE to enable access; FALSE to disable access.

p_schema
Name of the schema for the table or view.

p_object
Name of the table or view.

p_object_type
Type of the object: TABLE (default) or VIEW.

p_object_alias
Alias of the object.

p_auto_rest_auth
For an object, controls whether Oracle REST Data Services should require user authorization before allowing access to the Oracle REST Data Services metadata for this object.

Usage Notes

Only database users with the DBA role can enable/disable a objects other than their own.
ORDS.ENABLE_SCHEMA

Format

ORDS.ENABLE_SCHEMA
    p_enabled     BOOLEAN IN DEFAULT TRUE,
    p_schema      VARCHAR2(30) IN DEFAULT NULL,
    p_url_mapping_type VARCHAR2(10) IN DEFAULT 'BASE_PATH',
    p_url_mapping_pattern VARCHAR2(255) IN DEFAULT NULL,
    p_auto_rest_auth BOOLEAN IN DEFAULT NULL);

Description

ENABLE_SCHEMA enables Oracle REST Data Services to access the named schema.

Parameters

p_enabled
TRUE to enable Oracle REST Data Services access; FALSE to disable Oracle REST Data Services access.

p_schema
Name of the schema. If the p_schema parameter is omitted, then the current schema is enabled.

p_url_mapping_type
URL Mapping type: BASE_PATH or BASE_URL.

p_url_mapping_pattern
URL mapping pattern.

p_auto_rest_auth
For a schema, controls whether Oracle REST Data Services should require user authorization before allowing access to the Oracle REST Data Services metadata catalog of this schema.

Usage Notes

Only database users with the DBA role can enable/disable a schema other than their own.
ORDS.PUBLISH_MODULE

Format

```sql
ORDS.PUBLISH_MODULE(
    p_module_name  VARCHAR2(255) IN,
    p_status       VARCHAR2(30) IN DEFAULT 'PUBLISHED');
```

Description

PUBLISH_MODULE Change the publication status of an Oracle REST Data Services resource module.

Parameters

- **p_module_name**
  Current name of the RESTful service module. Case sensitive.

- **p_status**
  Publication status. Valid values: PUBLISHED (default) or NOT_PUBLISHED.
ORDS.RENAME_MODULE

Format

ORDS.RENAME_MODULE(
    p_module_name    VARCHAR2(255) IN,
    p_new_name    VARCHAR2(255) IN DEFAULT NULL,
    p_base_path      VARCHAR2(255) IN DEFEULT NULL);

Description

RENAME_MODULE lets you change the name or the base path, or both, of an Oracle REST Data Services resource module.

Parameters

p_module_name
Current name of the RESTful service module. Case sensitive.

p_new_name
New name to be assigned to the RESTful service module. Case sensitive. If this parameter is null, the name is not changed.

p_new_base_path
The base of the URI to be used to access this RESTful service. Example: hr/ means that all URIs starting with hr/ will be serviced by this resource module. If this parameter is null, the base path is not changed.
ORDS.RENAME_PRIVILEGE

Format

ORDS.RENAME_PRIVILEGE(
    p_name   VARCHAR2(255) IN,
    p_new_name  VARCHAR2(255) IN);

Description

RENAME_PRIVILEGE renames a privilege.

Parameters

  p_name
  Current name of the privilege.

  p_new_name
  New name to be assigned to the privilege.
ORDS.RENAME_ROLE

Format

ORDS.RENAME_ROLE(
    p_name     VARCHAR2(255) IN,
    p_new_name VARCHAR2(255) IN);

Description

RENAME_ROLE renames a role.

Parameters

p_name
Current name of the role.

p_new_name
New name to be assigned to the role.
ORDS.SET_MODULE_ORIGINS_ALLOWED

Format

ORDS.SET_MODULE_ORIGINS_ALLOWED
  p_module_name VARCHAR2(255) IN,
  p_origins_allowed VARCHAR2(255) IN;

Description

SET_MODULE_ORIGINS_ALLOWED configures the allowed origins for a resource module. Any existing allowed origins will be replaced.

Parameters

  p_module_name
  Name of the resource module.

  p_origins_allowed
  A comma-separated list of URL prefixes. If the list is empty, any existing origins are removed.

Usage Notes

To indicate no allowed origins for a resource module (and remove any existing allowed origins), specify an empty p_origins_allowed value.
ORDS.SET_URL_MAPPING

Format

ORDS.SET_URL_MAPPING
    p_schema        VARCHAR2(30) IN DEFAULT NULL,
    p_url_mapping_type  VARCHAR2(10) IN,
    p_url_mapping_pattern VARCHAR2(255) IN;

Description

SET_URL_MAPPING configures how the specified schema is mapped to request URLs.

Parameters

    p_schema
    Name of the schema to map.

    p_url_mapping_type
    URL Mapping type: BASE_PATH or BASE_URL.

    p_url_mapping_pattern
    URL mapping pattern.

Usage Notes

Only DBA users can update the mapping of a schema other than their own.
The OAUTH PL/SQL package contains procedures for implementing OAuth authentication using Oracle REST Data Services.

The Oracle REST Data Services PL/SQL API is explained in Section 3.9, "Using the Oracle REST Data Services PL/SQL API".
OAUTH.CREATE_CLIENT

Format

```
OAUTH.CREATE_CLIENT(
    p_name            VARCHAR2 IN,
    p_grant_type      VARCHAR2 IN,
    p_owner           VARCHAR2 IN DEFAULT NULL,
    p_description     VARCHAR2 IN DEFAULT NULL,
    p_allowed_origins VARCHAR2 IN DEFAULT NULL,
    p_redirect_uri    VARCHAR2 IN DEFAULT NULL,
    p_support_email   VARCHAR2 IN DEFAULT NULL,
    p_support_uri     VARCHAR2 IN DEFAULT NULL,
    p_privilege_names VARCHAR2 IN)
```

Description

Creates an OAuth client registration.

Parameters

**p_name**
Name for the client, displayed to the end user during the approval phase of three-legged OAuth. Must be unique.

**p_grant_type**
Must be one of `authorization_code`, `implicit`, or `client_credentials`.

**p_owner**
Name of the party that owns the client application.

**p_description**
Description of the purpose of the client, displayed to the end user during the approval phase of three-legged OAuth. May be null if `p_grant_type` is `client_credentials`; otherwise, must not be null.

**p_allowed_origins**
A comma-separated list of URL prefixes. If the list is empty, any existing origins are removed.

**p_redirect_uri**
Client-controlled URI to which redirect containing an OAuth access token or error will be sent. May be null if `p_grant_type` is `client_credentials`; otherwise, must not be null.

**p_support_email**
The email where end users can contact the client for support.

**p_support_uri**
The URI where end users can contact the client for support. Example: `http://www.myclientdomain.com/support/`

**p_privilege_names**
List of comma-separated privileges that the client wants to access.
OAUTH.DELETE_CLIENT

Format

OAUTH.DELETE_CLIENT(  
    p_name VARCHAR2 IN);

Description

Deletes an OAuth client registration (first format), or an OAuth client registration and all of the related privileges (second format).

Parameters

p_name
Name of the client registration to be deleted.
OAUTH.GRANT_CLIENT_ROLE

Format

OAUTH.GRANT_CLIENT_ROLE(
    p_client_name VARCHAR2 IN,
    p_role_name   VARCHAR2 IN);

Description

Grant an OAuth client the specified role, enabling clients performing two-legged OAuth to access Privileges requiring the role.

Parameters

p_client_name
Name of the OAuth client.

p_role_name
Name of the role to be granted.
OAUTH.RENAME_CLIENT

Format

OAUTH.RENAME_CLIENT(  
  p_name    VARCHAR2 IN,  
  p_new_name VARCHAR2 IN);  

Description

Renames a client.

Parameters

p_name
Current name for the client.

p_new_name
New name for the client.

Usage Notes

The client name is displayed to the end user during the approval phase of three-legged OAuth.
OAUTH.REVOKE_CLIENT_ROLE

Format

OAUTH.REVOKE_CLIENT_ROLE(
    p_client_name VARCHAR2 IN,
    p_role_name   VARCHAR2 IN);

Description

Revokes the specified role from an OAuth Client, preventing the client from accessing Privileges requiring the role via two-legged OAuth.

Parameters

p_client_name
Name of the OAuth client.

p_role_name
Name of the role to be revoked
OAUTH.UPDATE_CLIENT

Format

OAUTH.UPDATE_CLIENT(
    p_name             VARCHAR2 IN,
    p_owner            VARCHAR2 IN,
    p_description      VARCHAR2 IN,
    p_grant_type       VARCHAR2 IN,
    p_redirect_uri     VARCHAR2 IN,
    p_privilege_names  VARCHAR2 IN);

Description

Updates the client information.

Parameters

p_name
Name of the client.

p_owner
Name of the client.

p_description
description of the purpose of the client, displayed to the end user during the approval phase of three-legged OAuth. May be null if p_grant_type is client_credentials; otherwise, must not be null.

p_grant_type
Must be authorization_code, implicit, or client_credentials.

p_redirect_uri
Client-controlled URI to which a redirect containing the OAuth access token/error will be sent. May be null if p_grant_type is client_credentials; otherwise, must not be null.

p_privilege_names
Comma-separated names of the privileges that the client wishes to access.
The section describes the Oracle REST Data Services configuration files.

Topics:
- Locating Configuration Files
- Setting the Location of the Configuration Files
- Understanding the Configuration Folder Structure
- Understanding the Configuration File Format
- Understanding Configurable Parameters

A.1 Locating Configuration Files

Use the `configdir` command to display the current location of the configuration files:

```
java -jar ords.war configdir
```

If the configuration folder has not yet been configured, the message: `The config.dir setting is not set`, is displayed. If it has been configured, the current value of the setting is displayed.

A.2 Setting the Location of the Configuration Files

To change the location of the configuration folder use the `configdir` command:

```
java -jar ords.war configdir /path/to/config
```

Where:
- `/path/to/config` is the location where the configuration files are stored.

A.3 Understanding the Configuration Folder Structure

The configuration folder has the following structure:

```
./
|-- defaults.xml
`-- apex.properties
    `-- url-mapping.xml
```
Global settings that apply to all database connections are stored in `defaults.xml`. Settings specific to a particular database connection (for example, the default `apex` connection) are stored in `conf/<db-name>.xml`, where `<db-name>` is the name of the database connection.

If the database connection uses Oracle Application Express RESTful Services, the files with names including `_al.xml`, `_rt.xml`, and `_pu.xml` store the configuration for the `APEX_LISTENER`, `APEX_REST_PUBLIC_USER`, and `ORDS_PUBLIC_USER` database users, respectively.

If the database connection uses Oracle REST Data Services RESTful Services, the file `<db-name>_pu.xml` stores the configuration for the `ORDS_PUBLIC_USER` database user.

### A.4 Understanding the Configuration File Format

Configuration files use the standard Java XML properties file format, where each configuration setting contains a key and a corresponding value. The following is an example of a `defaults.xml` file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
  <entry key="db.connectionType">basic</entry>
  <entry key="db.hostname">localhost</entry>
  <entry key="db.port">1521</entry>
  <entry key="db.sid">orcl</entry>
  <entry key="jdbc.DriverType">thin</entry>
  <entry key="jdbc.InitialLimit">3</entry>
  <entry key="jdbc.MinLimit">1</entry>
  <entry key="jdbc.MaxLimit">10</entry>
  <entry key="jdbc.MaxStatementsLimit">10</entry>
  <entry key="jdbc.InactivityTimeout">1800</entry>
  <entry key="jdbc.statementTimeout">900</entry>
  <entry key="jdbc.MaxConnectionReuseCount">1000</entry>
</properties>
```

### A.4.1 Understanding the url-mapping.xml File Format

The `url-mapping.xml` file stores the rules that route requests to the appropriate database when more than one database is configured. The following is an example of a `url-mapping.xml` file:

```xml
<pool-config xmlns="http://xmlns.oracle.com/apex/pool-config">

... (rest of the file)
```

A-2 Oracle REST Data Services Installation, Configuration, and Development Guide
A.5 Understanding Configurable Parameters

Table A–1 lists editable parameters for the defaults.xml and (db-name).xml configuration files.

**Tip:** Oracle recommends users to use the Oracle REST Data Services command-line interface and Oracle SQL Developer Oracle REST Data Services Administration to edit the configuration files. For more information, see Section 1.3, "Installing and Configuring Oracle REST Data Services" and "Oracle REST Data Services Administration" in Oracle SQL Developer User’s Guide.

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>apex.docTable</td>
<td>string</td>
<td>Name of the document table used by Application Express.</td>
<td>MYDOCTABLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Defaults to FLOWS_FILES.WWV_FLOW_FILE_OBJECTS$.</td>
<td></td>
</tr>
<tr>
<td>apex.excel2collection</td>
<td>boolean</td>
<td>Indicate whether to place your Excel files into an Oracle Application Express collection.</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Supported values:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ true</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ false (default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>If value is true, then either apex.excel2collection.onecollection or</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>apex.excel2collection.useSheetName should be set to true.</td>
<td></td>
</tr>
<tr>
<td>apex.excel2collection.name</td>
<td>string</td>
<td>The name of the apex collection. The name is required if apex.excel2collection.onecollection is true.</td>
<td>mycollection</td>
</tr>
<tr>
<td>apex.excel2collection.onecollection</td>
<td>boolean</td>
<td>Indicate whether to put all Excel worksheets into a single collection.</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Supported values:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ true</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ false (default)</td>
<td></td>
</tr>
<tr>
<td>apex.excel2collection.useSheetName</td>
<td>boolean</td>
<td>Indicate whether to create a collection for each Excel worksheet, and uses each worksheet name for the corresponding collection name.</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Supported values:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ true</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ false (default)</td>
<td></td>
</tr>
</tbody>
</table>
### Understanding Configurable Parameters

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>cache.caching</td>
<td>boolean</td>
<td>Supported values:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td></td>
<td>false (default)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>For caching to be enabled, this must be set to true and the procedureNameList must have a procedure.</td>
<td></td>
</tr>
<tr>
<td>cache.directory</td>
<td>string</td>
<td>The directory location for the cache files.</td>
<td>C:\data\cachefiles</td>
</tr>
<tr>
<td>cache.duration</td>
<td>string</td>
<td>Supported values:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>days (default)</td>
<td>days</td>
</tr>
<tr>
<td></td>
<td></td>
<td>minutes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>hours</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Required for expire cache type.</td>
<td></td>
</tr>
<tr>
<td>cache.expiration</td>
<td>numeric</td>
<td>Required for expire cache type.</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Defaults to 7.</td>
<td></td>
</tr>
<tr>
<td>cache.maxEntries</td>
<td>numeric</td>
<td>Required for lru cache type.</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Defaults to 500.</td>
<td></td>
</tr>
<tr>
<td>cache.monitorInterval</td>
<td>numeric</td>
<td>Interval time is specified in minutes.</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If the cache type is expire, Oracle REST Data Services, checks the cache every ( NN ) minutes for files that have expired. For example, if the monitorInterval is 60, then it checks the cache every 60 minutes.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Defaults to 60.</td>
<td></td>
</tr>
<tr>
<td>cache.procedureNameList</td>
<td>string</td>
<td>Specify the procedure names to allow for caching of their files.</td>
<td>p, download_file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Procedure names can contain the wildcard characters asterisk (<em>) or question mark (?). Use an asterisk (</em>) to substitute zero or more characters and a question mark (?) to substitute for any one character.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Each procedure name must be separated by a comma.</td>
<td></td>
</tr>
<tr>
<td>cache.type</td>
<td>string</td>
<td>Supported values:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>expire</td>
<td>lru</td>
</tr>
<tr>
<td></td>
<td></td>
<td>lru (default)</td>
<td></td>
</tr>
<tr>
<td>db.connectionType</td>
<td>string</td>
<td>The type of connection. Supported values:</td>
<td>basic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>basic</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>tns</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>customurl</td>
<td></td>
</tr>
</tbody>
</table>

*Table A–1 (Cont.) Oracle REST Data Services Configuration Files Parameters*
<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>db.customURL</td>
<td>string</td>
<td>The JDBC URL connection to connect to the database.</td>
<td>jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=myhost)(PORT=1521)) (CONNECT_DATA=(SERVICE_NAME=ora111.example.com)))</td>
</tr>
<tr>
<td>db.hostname</td>
<td>string</td>
<td>The host system for the Oracle database.</td>
<td>myhostname</td>
</tr>
<tr>
<td>db.password</td>
<td>string</td>
<td>The password of the specified database user. Include an exclamation at the beginning of the password so that it can be stored encrypted.</td>
<td>!password4user</td>
</tr>
<tr>
<td>db.port</td>
<td>numeric</td>
<td>The database listener port.</td>
<td>1521</td>
</tr>
<tr>
<td>db.servicename</td>
<td>string</td>
<td>The network service name of the database.</td>
<td>ora111.example.com</td>
</tr>
<tr>
<td>db.serviceNameSuffix</td>
<td>string</td>
<td>Indicates that the pool points to a CDB, and that the PDBs connected to that CDB should be made addressable by Oracle REST Data Services (see Section D.3, &quot;Making All PDBs Addressable by Oracle REST Data Services (Pluggable Mapping)&quot;).</td>
<td>apex_pu</td>
</tr>
<tr>
<td>db.sid</td>
<td>string</td>
<td>The name of the database.</td>
<td>ora111</td>
</tr>
<tr>
<td>db.tnsAliasName</td>
<td>string</td>
<td>The TNS alias name that matches the name in the tnsnames.ora file.</td>
<td>MY_TNSALIAS</td>
</tr>
<tr>
<td>db.tnsDirectory</td>
<td>string</td>
<td>The directory location of your tnsnames.ora file.</td>
<td>C:\ORACLE\NETWORK\ADMIN</td>
</tr>
<tr>
<td>db.username</td>
<td>string</td>
<td>The name of the database user for the connection.</td>
<td>APEX_PUBLIC_USER</td>
</tr>
<tr>
<td>debug.debugger</td>
<td>boolean</td>
<td>Indicate whether to display debugging messages on the application server console. Supported values:</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ true</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ false (default)</td>
<td></td>
</tr>
<tr>
<td>debug.printDebugToScreen</td>
<td>boolean</td>
<td>Indicate whether to display error messages on the browser. Supported values:</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ true</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ false (default)</td>
<td></td>
</tr>
<tr>
<td>error.keepErrorMessages</td>
<td>boolean</td>
<td>Indicate whether to retain the error messages. Supported values:</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ true</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ false (default)</td>
<td></td>
</tr>
</tbody>
</table>
### Understanding Configurable Parameters

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>error.maxEntries</td>
<td>numeric</td>
<td>Specify the total number of error messages to retain.</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Defaults to 50.</td>
<td></td>
</tr>
<tr>
<td>icap.port</td>
<td>numeric</td>
<td>Specify the Internet Content Adaptation Protocol (ICAP) Port to virus scan files.</td>
<td>5555</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The icap.port is required to have a value.</td>
<td></td>
</tr>
<tr>
<td>icap.server</td>
<td>string</td>
<td>Specify the Internet Content Adaptation Protocol (ICAP) Server name to virus scan files.</td>
<td>servername</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The icap.server is required to have a value.</td>
<td></td>
</tr>
<tr>
<td>jdbc.DriverType</td>
<td>string</td>
<td>The JDBC driver type. Supported values:</td>
<td>thin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ thin</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ oci8</td>
<td></td>
</tr>
<tr>
<td>jdbc.InactivityTimeout</td>
<td>numeric</td>
<td>Specify how long an available connection can remain idle before it is closed. The inactivity connection timeout is in seconds.</td>
<td>1800</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Defaults to 1800.</td>
<td></td>
</tr>
<tr>
<td>jdbc.InitialLimit</td>
<td>numeric</td>
<td>Specify the initial size for the number of connections that will be created.</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Defaults to 3. (The default is low, and should probably be set higher in most production environments.)</td>
<td></td>
</tr>
<tr>
<td>jdbc.MaxConnectionReuseCount</td>
<td>numeric</td>
<td>Specify the maximum number of times to reuse a connection before it is discarded and replaced with a new connection.</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Defaults to 1000.</td>
<td></td>
</tr>
<tr>
<td>jdbc.MaxLimit</td>
<td>numeric</td>
<td>Specify the maximum number of connections.</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Defaults to 10. (Might be too low for some production environments.)</td>
<td></td>
</tr>
<tr>
<td>jdbc.maxRows</td>
<td>numeric</td>
<td>Specify the maximum number of rows that will be returned from a query when processing a RESTful service and that will be returned from a nested cursor in a result set. Affects all RESTful services generated through a SQL query, regardless of whether the resource is paginated.</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Defaults to 500.</td>
<td></td>
</tr>
<tr>
<td>jdbc.MaxStatementsLimit</td>
<td>numeric</td>
<td>Specify the maximum number of statements to cache for each connection.</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Defaults to 10.</td>
<td></td>
</tr>
<tr>
<td>jdbc.MinLimit</td>
<td>numeric</td>
<td>Specify the minimum number of connections.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Defaults to 1.</td>
<td></td>
</tr>
</tbody>
</table>
### Table A–1 (Cont.) Oracle REST Data Services Configuration Files Parameters

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>jdbc.statementTimeout</td>
<td>numeric</td>
<td>Specify how long a borrowed (in use) connection can remain unused before it is considered as abandoned and reclaimed. The abandoned connection timeout is in seconds.</td>
<td>900</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Defaults to 900.</td>
<td></td>
</tr>
<tr>
<td>log.logging</td>
<td>boolean</td>
<td>Indicate whether to retain the log messages.</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Supported values:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ true</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ false (default)</td>
<td></td>
</tr>
<tr>
<td>log.maxEntries</td>
<td>numeric</td>
<td>Specify the total number of log messages to retain.</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Defaults to 50.</td>
<td></td>
</tr>
<tr>
<td>log.procedure</td>
<td>boolean</td>
<td>Indicate whether procedures are to be logged.</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Supported values:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ true</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ false (default)</td>
<td></td>
</tr>
<tr>
<td>misc.defaultPage</td>
<td>string</td>
<td>The default page to display. The Oracle REST Data Services home page, apex, is commonly used.</td>
<td>apex</td>
</tr>
<tr>
<td>procedure.postProcess</td>
<td>string</td>
<td>Specify the procedure name(s) to execute after executing the procedure specified on the URL. Multiple procedure names must be separated by commas.</td>
<td>SCHEMA1.SUBMIT.RE QUEST, FINISHTASK</td>
</tr>
<tr>
<td>procedure.preProcess</td>
<td>string</td>
<td>Specify the procedure name(s) to execute prior to executing the procedure specified on the URL. Multiple procedure names must be separated by commas.</td>
<td>SCOTT.PREPROC1, INITIALIZE, PKG1.PROC</td>
</tr>
<tr>
<td>security.disableDefaultExclusionList</td>
<td>boolean</td>
<td>Supported values:</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ true</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>■ false (default)</td>
<td></td>
</tr>
<tr>
<td>security.exclusionList</td>
<td>string</td>
<td>Specify a pattern for procedures, packages, or schema names which are forbidden to be directly executed from a browser. Procedure names can contain the wildcard characters asterisk (<em>) or question mark (?). Use an asterisk (</em>) to substitute zero or more characters and a question mark (?) to substitute for any one character.</td>
<td>customer_account, bank*, employe?</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> Separate multiple patterns using commas.</td>
<td></td>
</tr>
</tbody>
</table>
### Table A–1  (Cont.) Oracle REST Data Services Configuration Files Parameters

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>security.inclusionList</td>
<td>string</td>
<td>Specify a pattern for procedures, packages, or schema names which are allowed to be directly executed from a browser. Procedure names can contain the wildcard characters asterisk (<em>) or question mark (?). Use an asterisk (</em>) to substitute zero or more characters and a question mark (?) to substitute for any one character. Note: Separate multiple patterns using commas.</td>
<td>apex, p, v, f, <code>wwv_*</code>, y*, c*</td>
</tr>
<tr>
<td>security.maxEntries</td>
<td>numeric</td>
<td>Specify the maximum cache size. Defaults to 2000.</td>
<td>2000</td>
</tr>
<tr>
<td>security.requestValidationFunction</td>
<td>string</td>
<td>Specify a validation function to determine if the requested procedure in the URL should be allowed or disallowed for processing. The function should return true if the procedure is allowed; otherwise, return false.</td>
<td>CHECK_VALID_PROCEDURE</td>
</tr>
<tr>
<td>security.verifySSL</td>
<td>boolean</td>
<td>Indicate whether HTTPS is available in your environment. Supported values: ■ true (default) ■ false If you change the value to false, see Section 1.3.6, &quot;Using OAuth2 in Non-HTTPS Environments&quot;.</td>
<td>true</td>
</tr>
<tr>
<td>soda.defaultLimit</td>
<td>string</td>
<td>When using the SODA REST API, specifies the default number of documents returned for a GET request on a collection when a limit is not specified in the URL. Must be a positive integer, or &quot;unlimited&quot; for no limit. Defaults to 100.</td>
<td>75</td>
</tr>
<tr>
<td>soda.maxLimit</td>
<td>string</td>
<td>When using the SODA REST API, specifies the maximum number of documents that will be returned for a GET request on a collection URL, regardless of any limit specified in the URL. Must be a positive integer, or &quot;unlimited&quot; for no limit. Defaults to 1000.</td>
<td>700</td>
</tr>
</tbody>
</table>
This appendix contains information on troubleshooting Oracle REST Data Services.

**Topics:**

- Enabling Debug Tracing
- Enabling Detailed Request Error Messages
- Configuring Application Express Static Resources with Oracle REST Data Services

**B.1 Enabling Debug Tracing**

To enable debug tracing, add the following setting to the Oracle REST Data Services configuration file named: `defaults.xml`:

```xml
<entry key="debug.debugger">true</entry>
```

When this setting is present in `defaults.xml`, detailed logging information that may help with problem diagnosis is appended to the Oracle REST Data Services log output. This setting should not be enabled on production systems due to the performance impact of outputting large amounts of data to the log.

**B.2 Enabling Detailed Request Error Messages**

To enable detailed request error messages, add the following setting to the Oracle REST Data Services configuration file named: `defaults.xml`:

```xml
<entry key="debug.printDebugToScreen">true</entry>
```

When this setting is present in `defaults.xml`, any request that produces an error response includes a detailed message, including a stack trace. This setting must not be enabled on production systems due to the risk of sensitive information being revealed to an attacker.

**B.3 Configuring Application Express Static Resources with Oracle REST Data Services**

When using Oracle REST Data Services, a blank page might be displayed when attempting to access an Oracle Application Express page, for example, when attempting to display `https://example/ords/`. This problem is caused by an improper configuration of Application Express static resources, which causes the JavaScript and CSS resources required by Application Express not to be found and the Application Express page not to render correctly.
The specific cause can be any of the following:

- Forgetting to ensure that the Application Express static images are located on the same server as the Oracle REST Data Services instance
- Forgetting to deploy i.war on WebLogic Server or GlassFish
- Specifying an incorrect path when using the `java -jar ords.war static` command to generate i.war
- Configuring Application Express to use a nondefault context path for static resources (/i) and not specifying the same context path (using the `--context-path` option) when using `java -jar ords.war static`
- Moving, renaming, or deleting the folder pointed to by i.war after deploying i.war
- When running in Standalone mode, entering an incorrect path (or not specifying a path) when prompted on the first run of Standalone mode
- When running in Standalone mode, entering an incorrect path with the `--static-images` option
- Upgrading to a new version of Application Express and forgetting to reconfigure and redeploy i.war to point to the static resources for the new Application Express version, or in Standalone mode forgetting to update the location by using the `--apex-images` option

To help in diagnosing the problem, you can try to access the `apex_version.txt` file. For example, if your Application Express deployment is located at https://example.com/ords/ and your static resources have been deployed at https://example.com/i/, use a browser to access the following URL:

https://example.com/i/apex_version.txt

If you get a 404 Not Found error, then check the preceding list of possible specific causes, including i.war not being deployed or not pointing to a folder containing Application Express static resources.

If a plain text file is displayed, it should contain text like the following:

```
Application Express Version:  4.2.1
```

Check that the version number matches the version of Application Express that is deployed on the database. If the numbers do not match, check if you have made an error mentioned in the last item in the preceding list of possible specific causes, because Oracle REST Data Services is not configured to use the correct version of the Application Express static resources to match the Application Express version in the database.

If you need help in solving the problem, check the information in this book about creating and deploying i.war for your environment, such as WebLogic Server or Glassfish.

You can also get detailed help on the static listener command by entering the following at a command prompt:

```
java -jar ords.war help static
```
This appendix explains an extended example that builds an image gallery service for storing and retrieving images. This tutorial uses Oracle Application Express.

To do this tutorial, you must be familiar with the concepts and techniques covered in Chapter 3, "Developing Oracle REST Data Services Applications".

Topics:
- Before You Begin
- Creating the Gallery Database Table
- Creating the Gallery RESTful Service Module
- Trying Out the Gallery RESTful Service
- Creating the Gallery Application
- Trying Out the Gallery Application
- Securing the Gallery RESTful Services
- Accessing the RESTful Services from a Third Party Application

C.1 Before You Begin

This section describes some common conventions used in this example as well as best practices regarding API entry points.

Topics:
- About URIs
- About Browser Support
- Creating an Application Express Workspace

C.1.1 About URIs

Throughout this example, URIs and URI Templates are referenced using an abbreviated form that omits the host name, context root and workspace path prefix. Consider the following example:

gallery/images/

To access this URI in your Web browser, you would use a URI in the following format:
Creating the Gallery Database Table

https://<host>:<port>/ords/<workspace>/gallery/images/

where
  ■  <host> is the host on which Oracle REST Data Services is running.
  ■  <port> is the port on which Oracle REST Data Services is listening.
  ■  /ords is the context root where Oracle REST Data Services is deployed.
  ■  /<workspace>/ is the workspace path prefix of the Oracle Application Express workspace where the RESTful Service is defined.

C.1.2 About Browser Support

This example uses many modern features defined in HTML5 and related specifications. It has only been tested in Mozilla Firefox and Google Chrome. It has not been tested in Microsoft Internet Explorer or on smart-phone/tablet web browsers. Please use recent versions of either Mozilla Firefox or Google Chrome for this example.

C.1.3 Creating an Application Express Workspace

To follow the instructions for creation the Gallery example application and related objects, first, create a new Oracle Application Express Workspace (in Full Development mode). See the Oracle Application Express Documentation for details on how to do this.

Call the workspace resteasy and call the administrator user of the workspace resteasy_admin. Ensure the resteasy_admin user is a member of the RESTful Services user group.

C.2 Creating the Gallery Database Table

To create the Gallery database table, follow these steps:

1. Log into the resteasy workspace.
2. Navigate to SQL Workshop and then SQL Commands.
3. Enter or copy and paste in the following SQL:

```sql
CREATE SEQUENCE GALLERY_SEQ
/
CREATE TABLE GALLERY (
  ID NUMBER NOT NULL ENABLE,
  TITLE VARCHAR2(1000) NOT NULL ENABLE,
  CONTENT_TYPE VARCHAR2(1000) NOT NULL ENABLE,
  IMAGE BLOB NOT NULL ENABLE,
  CONSTRAINT GALLERY_PK PRIMARY KEY (ID) ENABLE
)
/
CREATE OR REPLACE TRIGGER BI_GALLERY
before insert on GALLERY for each row
begin
  if :NEW.ID is null then
    select GALLERY_SEQ.nextval into :NEW.ID from sys.dual;
  end if;
end;
/
ALTER TRIGGER BI_GALLERY ENABLE
/```
C.3 Creating the Gallery RESTful Service Module

To create the Gallery RESTful services module, follow these steps:

1. Navigate to SQL Workshop and then RESTful Services.
2. Click Create on the right side, and enter the following information:
   - Name: gallery.example
   - URI Prefix: gallery/
   - URI Template: images/
   - Method: POST
   - Source: Enter or copy and paste in the following:
     
     ```
     declare
     image_id integer;
     begin
     insert into gallery (title, content_type, image)
     values (:title,:content_type,:body)
     returning id into image_id;
     :status := 201;
     :location := image_id;
     end;
     ```

3. Click Create Module.
4. Click the POST handler under images/
5. For Requires Secure Access, select No.
6. Click Create Parameter, and enter the following:
   - Name: Slug
   - Bind Variable Name: title
7. Click Create.
8. Click Create Parameter on the bottom right, and enter the following information:
   - Name: X-APEX-FORWARD
   - Bind Variable Name: location
   - Access Method: OUT
9. Click Create.
10. Click Create Parameter on the bottom right, and enter the following information:
    - Name: X-APEX-STATUS-CODE
    - Bind Variable Name: status
    - Access Method: OUT
    - Parameter Type: Integer
11. Click Create.

At this point you have created the module with a single service that can store new images. Next, add a service to display the list of stored images:

1. Navigate to SQL Workshop and then RESTful Services.
2. Click the module named gallery.example.
3. Click **Create Handler** under images/, and enter the following information:
   - **Method**: GET
   - **Source Type**: Feed
   - **Requires Secure Access**: No
   - **Source**: Enter or copy and paste in the following:
     
     ```
     select id,title,content_type from gallery order by id desc
     ```

4. Click **Create**.

   At this point you have created the service to store and list images. Next, add a service to display individual images:

   1. Navigate to **SQL Workshop** and then **RESTful Services**.
   2. Click the module named gallery.example.
   3. Click **Create Template** under gallery.example, and enter the following information:
      - **URI Template**: images/{id}

4. Click **Create**.

5. Click **Create Handler** under images/{id}, and enter the following information:
   - **Method**: GET
   - **Source Type**: Media Resource
   - **Requires Secure Access**: No
   - **Source**: Enter or copy and paste in the following:
     
     ```
     select content_type, image from gallery where id = :id
     ```

6. Click **Create**.

---

### C.4 Trying Out the Gallery RESTful Service

To try out the Gallery RESTful Service, follow these steps:

1. Navigate to **SQL Workshop** and then **RESTful Services**.
2. Click the module named gallery.example.
3. Click the **GET** handler located under images/.
4. Click **Test**.

   The following URI should be displayed in the browser:

   ```
   https://<host>:<port>/ords/resteasy/gallery/images/
   ```

   Content similar to the following should be displayed:

   ```
   {
   "next":
   "$ref":
   "http://localhost:8080/ords/resteasy/gallery/images/?page=1",
   "items":[]
   }
   ```
The content is a JSON document that lists the location of each image in the gallery, but since you have not yet added any images, the list (the \texttt{items[]} element) is empty.

The JSON has no extra white space to minimize its size, this can make it difficult to decipher, it is recommended to add a JSON viewing plugin to your browser to make viewing the JSON easier.

To create an Oracle Application Express application to enable users to add and view images in the gallery, see Section C.5, "Creating the Gallery Application".

C.5 Creating the Gallery Application

To create an Oracle Application Express application that uses the gallery RESTful Services, follow these steps:

1. Navigate to Application Builder.
2. Click Create.
3. Choose Database, then click Next.
4. Enter Image Gallery in the Name field, then click Next.
5. Click Create Application, and then Create Application again to confirm creation of the application.
6. Click page 1, Home.
7. Under Regions click the + (plus sign) icon to create a new region.
8. For Region Type, choose HTML and click Next, then click Next on the next page.
9. For Region Template, choose No Template.
10. For Title, enter Tasks, and click Next.
11. For Enter HTML Text Region Source, specify:
   \begin{verbatim}
   <a class="button" id="upload-btn">Upload Image</a>
   \end{verbatim}
12. Click Create Region.
13. Under Regions, click the + (plus sign) icon to create a new region.
14. For Region Type, choose HTML, and click Next, then Next again.
15. For Region Template, choose DIV Region with ID.
16. For Title, enter Images.
17. Click Create Region.
18. Click the Images region and click the Attributes tab.
19. For Static ID, enter images, and click Apply Changes.
20. Under Page, click the Edit icon, then click the JavaScript tab.
21. For Function and Global Variable Declaration, enter or copy and paste in the following:
   \begin{verbatim}
   var workspace_path_prefix = 'resteasy';
   var gallery_url = './' + workspace_path_prefix + '/gallery/images/';
   function uploadFiles(url, fileOrBlob, onload) {
     var name = 'unspecified';
     if (fileOrBlob['name']) {
       name = fileOrBlob.name;
   \end{verbatim}
Creating the Gallery Application

```javascript
function createUploader() {
    var $upload = $('div#uploader title="Image Upload"
        style="display:none">
        <form>
            <fieldset>
                <label for="file">File</label>
                <input type="file" name="file" id="file"
                    class="text ui-widget-content ui-corner-all" />
            </fieldset>
        </form>
    </div>');
    $(document.body).append($upload);
    $upload.dialog({
        autoOpen: false,
        modal: true,
        buttons: {
            "Upload": function() {
                var file = document.querySelector('input[type=file]');
                uploadFiles(gallery_url, file.files[0], function() {
                    $('#uploader').dialog("close");
                    getImages();
                });
            },
            "Cancel": function() {
                $('#uploader').dialog("close");
            }
        }
    });
    $('#upload-btn').click(function() {
        $('#uploader').dialog("open");
    });
}

function getImages() {
    var xhr = new XMLHttpRequest();
    xhr.open('GET', gallery_url);
    xhr.onload = function(e) {
        var data = JSON.parse(this.response);
        var $images = $('ol#image-list');
        for ( i in data.items ) {
            var item = data.items[i];
            var uri = item.uri['$ref'];
            var $image = $('li').append(
                '<a href="' + uri + '" title="' +
                item.title + '"><img src="graphics/' + uri + '">' +
                '</a>');</
                $images.append($image);
        }
        $('#images').append($images);
    }
    xhr.send();
}
```
22. For **Execute when Page Loads**, enter or copy and paste in the following:

```javascript
createUploader();
getImages();
```

23. **Click Apply Changes.**

24. **Under Page**, click the Edit icon, then click the **CSS** tab.

25. **For Inline**, enter or copy and paste in the following:

```css
a img { border:none; }
#images ol { margin: 1em auto; width: 100%; }
#images li { display: inline; }
#images a { background: #fff; display: inline; float: left; margin: 0 0 27px 30px; width: auto; padding: 10px 10px 15px; text-align: center; text-decoration: none; color: #333; font-size: 18px; -webkit-box-shadow: 0 3px 6px rgba(0,0,0,.25); -moz-box-shadow: 0 3px 6px rgba(0,0,0,.25); }
#images img { display: block; width: 190px; margin-bottom: 12px; }
label {width: 120px; margin-right: 0.625em; float: left; text-align: right; margin-bottom: 27px; padding-bottom: 10px; line-height: 18px; background: #fff; margin-bottom: 15px; text-decoration: none; color: #333; font-size: 18px; -webkit-box-shadow: 0 3px 6px rgba(0,0,0,.25); -moz-box-shadow: 0 3px 6px rgba(0,0,0,.25); background: -ms-linear-gradient(top, #3e779d, #65a9d7); background: -o-linear-gradient(top, #3e779d, #65a9d7); background: -webkit-linear-gradient(top, #3e779d, #65a9d7); background: -moz-linear-gradient(top, #3e779d, #65a9d7); padding: 5px 10px; border-radius: 8px; box-shadow: rgba(0,0,0,1) 0 1px 0; text-shadow: rgba(0,0,0,.4) 0 1px 0; color: white; font-size: 14px; text-decoration: none; vertical-align: middle; }
.button:hover { border-top-color: #28597a; background: #28597a; color: #ccc; cursor: pointer; }
```

Development Tutorial: Creating an Image Gallery  C-7
26. Click **Apply Changes**.

## C.6 Trying Out the Gallery Application

To try out the Gallery application, follow these steps:

1. Navigate to Application Builder.
2. Click Run beside the Image Gallery application.
3. Log in as the `resteasy_admin` user.
4. Click Upload Image.
5. Choose an image file (a JPEG or PNG file) and click Upload.

The application displays the uploaded image.

## C.7 Securing the Gallery RESTful Services

It is not wise to allow public access to the image uploading service, and it is probably not ideal to allow public access to the images in the gallery either. Therefore, you should protect access to the RESTful services.

RESTful Services support two kinds of authentication:

- **First Party Authentication.** This is authentication intended to be used by the party who created the RESTful service, enabling an Application Express application to easily consume a protected RESTful service. The application must be located with the RESTful service, that is, it must be located in the same Oracle Application Express workspace. The application must use the standard Oracle Application Express authentication.

- **Third Party Authentication.** This is authentication intended to be used by third party applications not related to the party who created the RESTful service. Third party authentication relies on the OAuth 2.0 protocol.

**Topics:**

- Protecting the RESTful Services
- Modifying the Application to Use First Party Authentication

### C.7.1 Protecting the RESTful Services

To protect the RESTful services, follow these steps:

1. Navigate to **SQL Workshop** and then **RESTful Services**.
2. Click **RESTful Service Privileges** in the section labeled **Tasks**.
3. Click **Create**, and enter the following:
   - **Name:** example.gallery
   - **Label:** Gallery Access
   - **Assigned Groups:** RESTful Services
Description: View and Post images in the Gallery

Protected Modules: gallery.example

4. Click Create.

To check that access to the RESTful Service is now restricted, follow these steps:

1. Navigate to SQL Workshop and then RESTful Services.
2. Click the module named gallery.example.
3. Click the GET handler located under images/.
4. Click Test.

The URI in the following format should be displayed in the browser:

https://<host>:<port>/ords/resteasy/gallery/images

An error page should be displayed with the error message:

401 Unauthorized.

This is the expected result, because a protected RESTful Service cannot be accessed unless proper credentials are provided. To add the required credentials to the request, see Section C.7.2, "Modifying the Application to Use First Party Authentication"

C.7.2 Modifying the Application to Use First Party Authentication

First Party Authentication relies on the cookie and user session established by the Application Express application, but Oracle REST Data Services needs additional information to enable it to verify the cookie. It needs to know the application ID and the current session ID. This information is always known to the Application Express application, and must be included with the request made to the RESTful service by adding the custom Apex-Session HTTP header to each request sent to a RESTful Service. The application ID and session ID are sent as the value of the header, separated from each other by a comma delimiter. For example:

GET /ords/resteasy/gallery/images/
Host: server.example.com
Apex-Session: 102,6028968452563

Sometimes it is not possible to include a custom header in the HTTP request. For example, when displaying an image in an HTML page using the <img> tag, an alternative mechanism is used for these scenarios. The application ID and session ID are included in a query parameter named _apex_session, which is added to the Request URI, which contains the application ID and session ID separated by a comma. For example:

<img src="graphics/101?_apex_session=102,6028968452563">

Note that this approach must only be used when it is not possible to use a custom header. Otherwise, this approach is discouraged because of the increased risk of the session ID being inadvertently stored or disclosed due to its inclusion in the URI.

To modify the application to add the first party authentication information to each request, follow these steps:

1. Navigate to Application Builder.
2. Click the Edit button beside the Image Gallery application.
3. Click the first page, named Home.
4. Under Page click the Edit icon, and click the JavaScript tab.

5. Add the following at the start of the Function and Global Variable Declaration field:

```javascript
function setApexSession(pathOrXhr) {
  var appId = $v('pFlowId');
  var sessionId = $v('pInstance');
  var apexSession = appId + ',' + sessionId;
  if ( typeof pathOrXhr === 'string' ) {
    var path = pathOrXhr;
    if ( path.indexOf('?') == -1 ) {
      path = path + '?_apex_session=' + apexSession;
    } else {
      path = path + '&_apex_session=' + apexSession;
    }
    return path;
  } else {
    var xhr = pathOrXhr;
    xhr.setRequestHeader('Apex-Session', apexSession);
    return xhr;
  }
}
```

6. This defines a JavaScript function named setApexSession() which will add the first party authentication information to an XMLHttpRequest object or a string containing a path.

Now you must modify the existing JavaScript code to add call this function when appropriate.

7. After the line reading `xhr.open('POST', url, true);`, add the following line:

```javascript
setApexSession(xhr);
```

8. After the line reading `xhr.open('GET', gallery_url);`, add the following line:

```javascript
setApexSession(xhr);
```

9. Change the line reading `var uri = item.uri['$ref'];` to:

```javascript
var uri = setApexSession(item.uri['$ref']);
```

10. Click Apply Changes.

11. Try running the application as before. It should work, because it is now providing the RESTful Services with the required authentication information.

**C.8 Accessing the RESTful Services from a Third Party Application**

If third parties want to consume and use the Gallery RESTful services, they must register the third party application in order to gain OAuth 2.0 credentials, which can then be used to initiate an interactive process by which users can authorize the third party application to access the RESTful Services on their behalf.

Once an application is registered, it can then acquire an access token. The access token must be provided with each request to a protected RESTful Service. Oracle REST Data Services verifies the access token before allowing access to the RESTful service.

OAuth 2.0 defines a number of different protocol flows that can be used by applications to acquire an access token. Oracle REST Data Services supports two of these protocol flows:
Accessing the RESTful Services from a Third Party Application

- **Authorization Code.** This flow is used when the third party application is able to keep its client credentials secure, for example, a third party website that is properly secured.

- **Implicit Grant.** This flow is used when the third party application cannot assure that its credentials would remain secret, for example, a JavaScript-based browser application or a native smartphone application.

The first step is to register the third party application. To demonstrate this, you will create a user representing the third party developer, and then use that user to register an application.

The steps in the related topics create a user in the RESTEASY workspace user repository and perform related actions.

---

**Note:** In addition to authenticating users defined in workspace user repositories, Oracle REST Data Services can also authenticate against any user repository accessible from WebLogic Server or GlassFish. For information, see Section 3.7, "Authenticating Against WebLogic Server and GlassFish User Repositories".

---

**Topics:**

- Creating the Third Party Developer User
- Registering the Third Party Application
- Acquiring an Access Token
- Using an Access Token
- About Browser Origins
- Configuring a RESTful Service for Cross Origin Resource Sharing
- Acquiring a Token using the Authorization Code Protocol Flow
- About Securing the Access Token

### C.8.1 Creating the Third Party Developer User

To create the third party developer user (the user account for the third party developer who wants to register an application to access the RESTful services), follow these steps:

1. Navigate to Administration.
2. Click Manage Users and Groups.
3. Click Create User, and enter the following information:
   - **Username:** 3rdparty_dev
   - **Email Address:** Email address for this developer user
   - **Password:** Password for this user
   - **User Groups:** OAuth 2.0 Client Developer
4. Click Create User.
## C.8.2 Registering the Third Party Application

To register the third party application to use the Implicit Grant OAuth 2.0 protocol flow, follow these steps:

1. Go to the following URI in your browser:
   
   ```
   https://server:port/ords/resteasy/ui/oauth2/clients/
   ```

2. Enter the credentials of the `3rdparty_dev` user created above, and click Sign In.

3. Click **Register Client**, and enter the following information:
   - **Name**: 3rd Party Gallery
   - **Description**: Demonstrates consuming the Gallery RESTful Service
   - **Response Type**: Token
   - **Support Email**: Desired email address
   - **Required Scopes**: Gallery Access

4. Click **Register**.

5. Click **3rd Party Gallery** in the list that appears on the next page.

6. Note the values of the Client Identifier and the Authorization URI fields.

## C.8.3 Acquiring an Access Token

To acquire an access token, a user must be prompted to approve access. To initiate the approval process, direct the user to the approval page using the following URI:

```
   client_id=CLIENT_IDENTIFIER&
   state=STATE
```

where:

- **CLIENT_IDENTIFIER** is the Client Identifier assigned to the application when it was registered.
- **STATE** is a unique value generated by the application used to prevent Cross Site Request Forgery (CSRF) attacks.

Note the following about the Oracle REST Data Services OAuth 2.0 implementation:

- The OAuth 2.0 specification allows two optional parameters to be supplied in the above request:
  - **redirect_uri**: Identifies the location where the authorization server will redirect back to after the user has approved/denied access.
  - **scope**: Identifies the RESTful Service Privileges that the client wishes to access.

Oracle REST Data Services does not support either of these parameters: both of these values are specified when the client is registered, so it would be redundant to repeat them here. Any values supplied for these parameters will be ignored.

- The OAuth 2.0 specification recommends the use of the **state** parameter, but Oracle REST Data Services requires the use of the parameter because of its importance in helping to prevent CSRF attacks.

- The response type is also specified when the application is registered, and thus the **response_type** parameter is also redundant; however, the OAuth 2.0 specification
states the parameter is always required, so it must be included. It is an error if the
response_type value differs from the registered response type.

When the preceding URI is accessed in a browser, the user is prompted to sign on, and
then prompted to review the application’s request for access and choose whether to
approve or deny access.

If the user approves the request, then the browser will be redirected back to the
registered redirect URI, and the access token will be encoded in the fragment portion
of the URI:

https://server:port/i/oauthdemo/gallery.html#token_type=bearer&
access_token=ACCESS_TOKEN&
expires_in=TOKEN_LIFETIME&
state=STATE

where:

- **ACCESS_TOKEN** is the unique, unguessable access token assigned to the current user
  session, and which must be provided with subsequent requests to the RESTful
  service.
- **TOKEN_LIFETIME** is the number of seconds for which the access token is valid.
- **STATE** is the unique value supplied by the application at the start of the
  authorization flow. If the returned state value does not match the initial state
  value, then there is an error condition, and the access token must not be used,
  because it is possible an attacker is attempting to subvert the authorization process
  through a CSRF attack.

If the user denies the request, or the user is not authorized to access the RESTful
Service, the browser will be redirected back to the registered redirect URI, and an error
message will be encoded in the fragment portion of the URI:

https://server:port/i/oauthdemo/gallery.html#error=access_denied&state=STATE

where:

- **error=access_denied** informs the client that the user is not authorized to access
  the RESTful Service, or chose not to approve access to the application.
- **STATE** is the unique value supplied by the application at the start of the
  authorization flow. If the returned state value does not match the initial state
  value, then there is an error condition, the client should ignore this response. It is
  possible an attacker is attempting to subvert the authorization process via a CSRF
  attack.

**C.8.4 Using an Access Token**

After the application has acquired an access token, the access token must be included
with each request made to the protected RESTful service. To do this, an Authorization
header is added to the HTTP request, with the following syntax:

Authorization: Bearer ACCESS_TOKEN

where:

- **ACCESS_TOKEN** is the access token value.

For example, a JavaScript-based browser application might invoke the Gallery service
as follows:

```javascript
var accessToken = ... /* initialize with the value of the access token */
```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'https://server:port/ords/resteasy/gallery/images/',true);
/* Add the Access Token to the request */
xhr.setRequestHeader('Authorization', 'Bearer ' + accessToken);
xhr.onload = function(e) {
    /* logic to process the returned JSON document */
    ...
};
xhr.send();

The preceding example uses the XMLHttpRequest.setRequestHeader(name,value) function to add the Authorization header to the HTTP request. If the access token is valid, then the server will respond with a JSON document listing the images in the gallery.

C.8.5 About Browser Origins

One of the key security concepts of web browsers is the Same Origin Policy, which permits scripts running on pages originating from the same web site (an Origin) to access each other's data with no restrictions, but prevents access to data originating from other web sites.

An origin is defined by the protocol, host name and port of a web-site. For example https://example.com is one origin and https://another.example.com is a different origin, because the host name differs. Similarly, http://example.com is a different origin than https://example.com because the protocol differs. Finally, http://example.com is a different origin from http://example.com:8080 because the port differs.

For example, if a third party client of the Gallery RESTful service is located at:
https://thirdparty.com/gallery.html

and the Gallery RESTful service is located at:
https://example.com/ords/resteasy/gallery/images/

then the Same Origin Policy will prevent gallery.html making an XMLHttpRequest to https://example.com/ords/resteasy/gallery/images/, because scripts in the https://thirdparty.com origin can only access data from that same origin, and https://example.com is clearly a different origin.

This is proper if the authors of https://example.com do not trust the authors of https://thirdparty.com. However, if the authors do have reason to trust each other, then the Same Origin Policy is too restrictive. Fortunately, a protocol called Cross Origin Resource Sharing (CORS), provides a means for https://example.com to inform the web browser that it trusts https://thirdparty.com and thus to instruct the browser to permit gallery.html to make an XMLHttpRequest to https://example.com/ords/resteasy/gallery/images/.

C.8.6 Configuring a RESTful Service for Cross Origin Resource Sharing

To configure a RESTful service for Cross Origin Resource Sharing, follow these steps:

1. Navigate to SQL Workshop and then RESTful Services.
2. Click the module named gallery.example.
3. For Origins Allowed, enter the origins that are permitted to access the RESTful service (origins are separated by a comma).
4. Press Apply Changes

C.8.7 Acquiring a Token using the Authorization Code Protocol Flow

Other sections have explained acquiring an access token using the OAuth 2.0 Implicit protocol flow. This section explains how to do the same using the Authorization Code protocol flow. The process is slightly more involved than for the Implicit protocol flow, because it requires exchanging an authorization code for an access token.

This section will mimic this exchange process using cURL.

Topics:
- Registering the Client Application
- Acquiring an Authorization Code
- Exchanging an Authorization Code for an Access Token
- Extending OAuth 2.0 Session Duration

C.8.7.1 Registering the Client Application

To register the client, follow these steps:

1. Go to the following URI in your browser:
   https://server:port/ords/resteasy/ui/oauth2/clients/
2. Enter the credentials of the 3rdparty_dev user, and click Sign In.
3. Click Register Client, and enter the following information:
   - Name: Another Gallery
   - Description: Demonstrates using the Authorization Code OAuth 2.0 Protocol Flow
   - Response Type: Code
   - Redirect URI: https://gallery.example.demo
   - Support Email: any desired email address
   - Required Scopes: Gallery Access
4. Click Register.
5. Click 3rd Party Gallery in the list that appears on the next page.
6. Note the values of the Client Identifier, Client Secret, and the Authorization URI fields.

C.8.7.2 Acquiring an Authorization Code

The first step in the Authorization Code protocol flow is to acquire an authorization code. An authorization code is a short lived token that when presented along with the application's client identifier and secret can be exchanged for an access token.

To acquire an access token, the user must be prompted to approve access. To initiate the approval process, direct the user to the approval page using a URI in the following format:

   client_id=CLIENT_IDENTIFIER&
   state=STATE
where:

- **CLIENT_IDENTIFIER** is the Client Identifier assigned to the application when it was registered.
- **STATE** is a unique value generated by the application used to prevent Cross Site Request Forgery (CSRF) attacks.

If the user approves the request, then the browser will be redirected back to the registered redirect URI, and the access token will be encoded in the query string portion of the URI:

https://gallery.example.demo?code=AUTHORIZATION_CODE&state=STATE

where:

- **AUTHORIZATION_CODE** is the authorization code value.
- **STATE** is the unique value supplied by the application at the start of the authorization flow. If the returned state value does not match the initial state value, then there is an error condition, the authorization code must not be used. It is possible an attacker is attempting to subvert the authorization process via a CSRF attack.

Because the registered https://gallery.example.demo redirect URI does not exist, the browser will report a server not found error, but for the purposes of this example, this does not matter, because you can still see the authorization code value encoded in the URI. Note the value of the code parameter, because it will be used in Section C.8.7.3, "Exchanging an Authorization Code for an Access Token".

### C.8.7.3 Exchanging an Authorization Code for an Access Token

In Section C.8.7.2, "Acquiring an Authorization Code" you acquired an authorization code. In this section you will use cURL to exchange the authorization code for an access token.

To exchange an authorization code the application must make an HTTP request to the Oracle REST Data Services OAuth 2.0 token endpoint, providing the authorization code and its client identifier and secret. If the credentials are correct, Oracle REST Data Services responds with a JSON document containing the access token. Note that the application makes the HTTP request from its server side (where the client identifier and secret are securely stored) directly to Oracle REST Data Services; the web-browser is not involved at all in this step of the protocol flow.

Use a cURL command in the following format to exchange the authorization code for an access token:

```bash
curl -i -d "grant_type=authorization_code&code=AUTHORIZATION_CODE" \   --user CLIENT_IDENTIFIER:CLIENT_SECRET \   https://server:port/ords/resteasy/oauth2/token
```

where:

- **AUTHORIZATION_CODE** is the authorization code value (which was encoded in the code parameter of the query string in the redirect URI in the previous section).
- **CLIENT_IDENTIFIER** is the client identifier value.
- **CLIENT_SECRET** is the client secret value.

CURL translates the above commands into an HTTP request like the following:

```
POST /ords/resteasy/oauth2/token HTTP/1.1
Authorization: Basic Q0xJRU5UX01ERU5USUZJRVI6Q0xJR5UX1NFQ1JFVAAA
```
Accessing the RESTful Services from a Third Party Application

Development Tutorial: Creating an Image Gallery  C-17

Host: server:port
Accept: */*
Content-Length: 59
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&code=AUTHORIZATION_CODE

where:

- The request is an HTTP POST to the oauth2/token OAuth 2.0 token endpoint.
- The Authorization header uses the HTTP BASIC authentication protocol to encode the client identifier and secret to assert the application’s identity.
- The Content-Type of the request is form data (application/x-www-form-urlencoded) and the content of the request is the form data asserting the OAuth 2.0 token grant type and the OAuth 2.0 authorization code value.

The preceding HTTP request will produce a response like the following:

HTTP/1.1 200 OK
ETag: "...
Content-Type: application/json

{
  "access_token": "04tss-gM35uOeQzR_2ve4Q..",
  "token_type": "bearer",
  "expires_in": 3600,
  "refresh_token": "UX4FVhPFJ16GokvTXYw0A.."
}

The response is a JSON document containing the access token along with a refresh token (more about refresh tokens in Section C.8.7.4, "Extending OAuth 2.0 Session Duration"). After the application has acquired an access token, the access token must be included with each request made to the protected RESTful Service. To do this an Authorization header is added to the HTTP request, with the following syntax:

Authorization: Bearer ACCESS_TOKEN

C.8.7.4 Extending OAuth 2.0 Session Duration

To extend the lifetime of an OAuth 2.0 session, a refresh token can be exchanged for a new access token with a new expiration time. Note that refresh tokens are only issued for the Authorization Code protocol flow.

The application makes a similar request to that used to exchange an authorization code for an access token. Use a cURL command in the following format to exchange the refresh token for an access token:

curl -i -d "grant_type=refresh_token&refresh_token=REFRESH_TOKEN" \
--user CLIENT_IDENTIFER:CLIENT_SECRET \nhttps://server:port/ords/resteasy/oauth2/token

where:

- REFRESH_TOKEN is the refresh token value returned when the access token was initially issued.
- CLIENT_IDENTIFER is the client identifier value.
- CLIENT_SECRET is the client secret value.

cURL translates the above commands into an HTTP request like the following:
POST /ords/resteasy/oauth2/token HTTP/1.1
Authorization: Basic Q0xJRU5UX01ERU5USUZJRVI6Q0xJRU5UX1NFQlJFVA==
Host: server:port
Accept: */*
Content-Length: 53
Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token&refresh_token=REFRESH_TOKEN

where:
- The request is an HTTP POST to the oauth2/token OAuth 2.0 token endpoint.
- The Authorization header uses the HTTP BASIC authentication protocol to encode the client identifier and secret to assert the application’s identity.
- The Content-Type of the request is form data (application/x-www-form-urlencoded) and the content of the request is the form data asserting the OAuth 2.0 token grant type and the refresh token value.

The preceding HTTP request will produce a response like the following:

HTTP/1.1 200 OK
ETag: "..."
Content-Type: application/json

{  
  'access_token': 'hECH_Fc7os2KtXT4pDfkzw..',  
  'token_type': 'bearer',  
  'expires_in': 3600,  
  'refresh_token': '-7OBQKc_gUQG932HCi08Hg..'  
}

The response is a JSON document containing the new access token along with a new refresh token. The existing access token and refresh token are invalidated, and any attempt to access a service using the old access token will fail.

C.8.8 About Securing the Access Token

In OAuth 2.0 the access token is the sole credential required to provide access to a protected service. It is, therefore, essential to keep the access token secure. Follow these guidelines to help keep the token secure:

- It is strongly recommended to use HTTPS for all protected RESTful Services. This prevents snooping attacks where an attacker may be able to steal access tokens by eavesdropping on insecure channels. It also prevents attackers from viewing the sensitive data that may be present in the payload of the requests.

- Ensure that the client application is not located in a browser origin with other applications or scripts that cannot be trusted. For example assume that user Alice has a client application hosted at the following location:

  https://sharedhosting.com/alice/application

  If another user (such as Fred) is also able to host his application in the same origin, for example, at:

  https://sharedhosting.com/fred/trouble

  then it will be easy for /fred/trouble to steal any access token acquired by /alice/application, because they share the same origin.
https://sharedhost.com, and thus the browser will not prevent either application from accessing the other's data.

To protect against this scenario, Alice's application must be deployed in its own origin, for example:

https://alice.sharedhosting.com/application

or:

https://application.alice.sharedhosting.com

or:

https://aliceapp.com
Using the Multitenant Architecture with Oracle REST Data Services

This section outlines the installation choices and different scenarios associated with copying and moving pluggable databases introduced by the Oracle Database 12c multitenant architecture with respect to Oracle REST Data Services.

- Understanding the Installation Choices
- Installing Oracle REST Data Services into a CDB
- Making All PDBs Addressable by Oracle REST Data Services (Pluggable Mapping)
- Uninstalling Oracle REST Data Services from a CDB

D.1 Understanding the Installation Choices

Oracle Database 12c Release 1 (12.1) introduces the multitenant architecture. This database architecture has a multitenant container database (CDB) that includes a root container, CDB$ROOT, a seed database, PDB$SEED, and multiple pluggable databases (PDBs). A PDB appears to users and applications as if it were a non-CDB. Each PDB is equivalent to a separate database instance in Oracle Database Release 11g.

The root container, CDB$ROOT, holds common objects that are accessible to every PDB utilizing metadata links or object links. The seed database, PDB$SEED, is used when creating a new PDB to seed the new pluggable database. The key benefit of the Oracle Database 12c multitenant architecture is that the database resources, such as CPU and memory, can be shared across all of the PDBs. This architecture also enables many databases to be treated as one for tasks such as upgrades or patches, and backups.

You can install Oracle REST Data Services into one or more pluggable databases PDBs in a multitenant database or into the container database (CDB). The installation choices are as follows:

- If you want the same version of Oracle REST Data Services available in all the PDBs, then install it into the CDB. The rest of the instructions in this topic refer to installing into the CDB.
- If you want only some PDBs to be able to use Oracle REST Data Services, or if you want different PDBs to use different versions of Oracle REST Data Services, then install into the desired PDBs. (Use the same procedure as for a non-CDB.)

When Oracle REST Data Services is installed into a CDB, it is installed in the root container, the seed container, and any existing PDBs. The root container (CDB$ROOT) includes the ORDS_METADATA schema to store the common database objects for Oracle REST Data Services packages, functions, procedures, and views. It also includes the Oracle REST Data Services public user (ORDS_PUBLIC_USER).
D.2 Installing Oracle REST Data Services into a CDB

If you want to have all PDBs in a multitenant environment to use the same Oracle REST Data Services release and patch set, install into the CDB. (This option will not allow you to have different releases of Oracle REST Data Services in different PDBs.)

Before installing into the a CDB:

- Ensure that the PDBs are open (not mounted/closed) in read/write mode (except for PDB$SEED, which remains in read-only mode). See "Modifying the Open Mode of PDBs" in Oracle Database Administrator’s Guide.

- Ensure that the default and temporary tablespaces to be used by the ORDS_METADATA schema and the ORDS_PUBLIC_USER user exist and that you know the tablespace names. The installation procedure creates those users, but it does not create the tablespaces.

Note that ORDS_METADATA and ORDS_PUBLIC_USER are also installed in the seed container, and that the default and temporary tables will have to exist in PDB$SEED. If these tablespace do not already exist there, then you will have to create the tablespaces in PDB$SEED; see "Running Oracle-Supplied SQL Scripts in a CDB" in Oracle Database Administrator’s Guide.

To install Oracle REST Data Services into a CDB, follow these steps.

1. Go to the folder into which you unzipped the Oracle REST Data Services installation kit.

2. Extract the installation scripts by entering this command:
   
   java -jar ords.war ords-scripts

3. Go to the folder containing the installation core scripts (scripts\install\core\).

4. Create a folder to contain Oracle REST Data Services log files (for example, d:\scripts\logs), or skip this step if you plan to use an existing folder for these log files.

5. Using SQL*Plus connect to the root container as SYS AS SYSDBA.

6. Verify that the container name is CDB$ROOT by executing the following statement:
   
   show conn_name

7. Execute the ords_manual_install.sql script. For example:
   
   @ords_manual_install SYSAUX TEMP d:\scripts\logs\

   Where the arguments specify:

   - ORDS_METADATA default tablespace (must already exist)
- ORDS_METADATA temporary tablespace (must already exist)
- Location for the Oracle REST Data Services log files (must already exist; specifying the trailing slash)

You will be prompted for the ORDS_PUBLIC_USER password.
You will be also prompted for the ORDS_PUBLIC_USER default and temporary tablespaces.

8. Verify that the installation was successful (a "completed successfully" message is displayed).
   If an error occurred, view the log files.

9. Create the Oracle REST Data Services configuration files, as follows:
   a. Go to the folder into which you unzipped the Oracle REST Data Services installation kit.
   b. Enter the following command:
      
      java -jar ords.war install advanced
   c. When prompted, enter the database connection information for your CDB. (To use the pluggable mapping feature, see Section D.3, "Making All PDBs Addressable by Oracle REST Data Services (Pluggable Mapping)"). For example:
      
      Enter the name of the database server [localhost]:
      Enter the database listen port [1521]:
      Enter 1 to specify the database service name, or 2 to specify the database SID [1]:
      Enter the database service name: cdb.example.com
   d. Verify the Oracle REST Data Services installation:
      Enter 1 if you want to verify/install Oracle REST Data Services schema or 2 to skip this step [1]:
   e. Enter and confirm the ORDS_PUBLIC_USER password:
      Enter the database password for ORDS_PUBLIC_USER:
      Confirm password:
   f. When prompted, enter additional information as needed. See Section 1.3.4.2, "Advanced Installation Using Command-Line Prompts".

D.3 Making All PDBs Addressable by Oracle REST Data Services (Pluggable Mapping)

Pluggable mapping refers to the ability to make all PDBs in a CDB addressable by Oracle REST Data Services. To use this feature, follow the instructions in this topic.

If the Oracle REST Data Services configuration file (see Appendix A, "About the Oracle REST Data Services Configuration Files") includes the db.serviceNameSuffix parameter, this indicates that the Oracle REST Data Services pool points to a CDB, and that the PDBs connected to that CDB should be made addressable by Oracle REST Data Services.

The value of the db.serviceNameSuffix parameter must match the value of the DB_DOMAIN database initialization parameter, and it must start with a period (.). To set the value of the db.serviceNameSuffix parameter:
1. In SQL*Plus, connect to the root as a user with SYSDBA privileges.

2. Check the value of the DB_DOMAIN database initialization parameter.

   SQL> show parameter DB_DOMAIN

3. Exit SQL*Plus.

   SQL> exit

4. If the DB_DOMAIN value was not empty, then on the command line enter the command to create the key and value for the db.serviceNameSuffix parameter and its DB_DOMAIN. This will be used to add this entry to the ORDS configuration file.

   echo db.serviceNameSuffix=.value-of-DB_DOMAIN > snsuffix.properties

   For example, if DB_DOMAIN is set to example.com, enter the following:

   echo db.serviceNameSuffix=.example.com > snsuffix.properties

5. If the db.serviceNameSuffix parameter value is not defined, enter a command in the following format to add an entry to the configuration file:

   java -jar ords.war set-properties --conf pool-name snsuffix.properties

   Where pool-name is one of the following:
   - poolName for a PL/SQL Gateway configuration
   - poolName_ru for an Oracle REST Data Services RESTful Services configuration
   - poolName_rt for an Application Express RESTful Services configuration

   Example 1: You want to make PDBs in a CDB addressable for your PL/SQL Gateway, and your pool name is apex. Enter the following command:

   java -jar ords.war set-properties --conf apex snsuffix.properties

   For example, if the database pointed to by apex has a DB_DOMAIN value of example.com and contains the two PDBs pdb1.example.com and pdb2.example.com, the first PDB will be mapped to URLs whose path starts with /ords/pdb1/, and the second PDB will be mapped to URLs whose path starts with /ords/pdb2/.

   Example 2: You want to make PDBs in a CDB addressable for your Oracle REST Data Services RESTful Services, and your pool name is apex_pu. Enter the following command:

   java -jar ords.war set-properties --conf apex_pu snsuffix.properties

   Example 3: You want to make PDBs in a CDB addressable for your Application Express RESTful Services and your pool name is apex_rt. Enter the following command:

   java -jar ords.war set-properties --conf apex_rt snsuffix.properties

   Example 4: You want to make PDBs in a CDB addressable globally (for the situations in the three preceding examples). Specify defaults by entering the following command:

   java -jar ords.war set-properties --conf defaults snsuffix.properties
D.4 Uninstalling Oracle REST Data Services from a CDB

To uninstall Oracle REST Data Services from a CDB, follow these steps.

1. Extract the Oracle REST Data Services scripts, as explained in Section 1.3.4.3, "Extracting the Scripts to Install and Uninstall Oracle REST Data Services".

2. Go to the folder where you extracted the scripts.

3. Using SQL*Plus, connect to the root container as SYS AS SYSDBA.

4. Verify that the container name is CDB$ROOT by executing the following statement:
   
   ```sql
   show conn_name
   ```

5. Execute the `ords_manual_uninstall.sql` script. For example:
   
   ```sql
   @ords_manual_uninstall d:\scripts\logs
   ```
   
   Where the argument specifies the location for the Oracle REST Data Services log files (must already exist).

6. Verify that the uninstall operation was successful (a “completed successfully” message is displayed).

   If an error occurred, view the log files.
This tutorial is designed to let you get started quickly developing RESTful services using Oracle REST Data Services. If you go through this tutorial before reading Chapter 3, "Developing Oracle REST Data Services Applications", you should later read that chapter to understand the main concepts and techniques presented there.

This tutorial is based on the following document that is included in the Oracle REST Data Services installation kit: <install-directory>\examples\getting-started\index.html. That document is among several resources supplied to help you become productive quickly, as explained in Section 3.2.3, "Getting Started" Documents Included in Installation.

Before you perform the actions in this tutorial, note the following prerequisites and recommendations:

- Ensure that you have installed Oracle REST Data Services and configured it to connect to an Oracle database.
- Ensure that you have installed Oracle SQL Developer 4.1 or later in order to be able to edit RESTful services.
- It is strongly recommended that you install a browser extension that enables you to view JSON in the web browser. Recommended extensions:
  - For Google Chrome: JSON Formatter (https://chrome.google.com/webstore/detail/json-formatter/bcjindccaaagfpapjjaapmmgkkhgoa)
  - For Mozilla Firefox: JSON View (https://addons.mozilla.org/en-US/firefox/addon/jsonview/)

The examples in this tutorial assume the following:

- Oracle REST Data Services has been installed and configured, and is running in standalone mode on the following server, port, and context path: localhost:8080/ords/
- Oracle REST Data Services is configured to connect as its default connection to an Oracle database listening on localhost:1521, and the database has a service name of orcl.

This "getting started" tutorial has the following major steps:

1. REST-Enable a Database Table
2. Create a RESTful Service from a SQL Query
3. Protect Resources
4. Register an OAuth Client Application
E.1 REST-Enable a Database Table

To enable a table for REST access, follow these steps.

**Tip:** It is recommended that you follow the steps as closely as possible, including using the specified names for schemas and database objects. After you have successfully completed the tutorial using this approach, feel free to try it again using other values if you wish.

1. Create a database schema named `ordstest` for use in trying out RESTful services. In SQL Developer, connect to the database as `sys` and enter the following in the SQL Worksheet:

   ```sql
   CREATE USER ordstest IDENTIFIED BY <password>;
   GRANT "CONNECT" TO ordstest;
   GRANT "RESOURCE" TO ordstest;
   GRANT UNLIMITED TABLESPACE TO ordstest
   ```

2. Connect to the `ordstest` schema. In SQL Developer create a connection to the `ordstest` schema, connect to it, and open a SQL worksheet.

3. Create a database table. For example, enter the following in the SQL Worksheet to create an example table named `EMP`:

   ```sql
   CREATE TABLE EMP (  
   EMPNO NUMBER(4,0),  
   ENAME VARCHAR2(10 BYTE),  
   JOB VARCHAR2(9 BYTE),  
   MGR NUMBER(4,0),  
   HIREDATE DATE,  
   SAL NUMBER(7,2),  
   COMM NUMBER(7,2),  
   DEPTNO NUMBER(2,0),  
   CONSTRAINT PK_EMP PRIMARY KEY (EMPNO)
   );
   ```

4. Insert some sample data into the table. For example:

   ```sql
   Insert into EMP (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values (7369,'SMITH','CLERK',7902,to_date('17-DEC-80','DD-MON-RR'),800,null,20);
   Insert into EMP (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values (7499,'ALLEN','SALESMAN',7698,to_date('20-FEB-81','DD-MON-RR'),1600,300,30);
   Insert into EMP (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values (7521,'WARD','SALESMAN',7698,to_date('22-FEB-81','DD-MON-RR'),1250,500,30);
   Insert into EMP (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values (7566,'JONES','MANAGER',7839,to_date('02-APR-81','DD-MON-RR'),2975,null,20);
   Insert into EMP (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values (7654,'MARTIN','SALESMAN',7698,to_date('28-SEP-81','DD-MON-RR'),1250,1400,30);
   Insert into EMP (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values (7698,'BLAKE','MANAGER',7839,to_date('01-MAY-81','DD-MON-RR'),2850,null,30);
   Insert into EMP (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values (7782,'CLARK','MANAGER',7839,to_date('09-JUN-81','DD-MON-RR'),2450,null,10);
   Insert into EMP (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values (7788,'SCOTT','ANALYST',7566,to_date('19-APR-87','DD-MON-RR'),3000,null,20);
   Insert into EMP (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values (7839,'KING','PRESIDENT',null,to_date('17-NOV-81','DD-MON-RR'),5000,null,10);
   Insert into EMP (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values (7844,'TURNER','SALESMAN',7698,to_date('08-SEP-81','DD-MON-RR'),1500,0,30);
   Insert into EMP (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values (7876,'ADAMS','CLERK',7788,to_date('23-MAY-87','DD-MON-RR'),1100,null,20);
   ```
5. Enable the schema of the EMP table for REST. In SQL Developer, right-click the ordstest connection, and select REST Services > Enable RESTful Services to display the following wizard page:


- **Enable schema**: Enable this option.
- **Schema alias**: Accept ordstest for the schema alias.
- **Authorization required**: For simplicity, this tutorial does not require authorisation, so disable this option.

Click Next.

6. On the RESTful Summary page of the wizard, click Finish.

7. Enable the EMP table for REST. In SQL Developer, right-click EMP table in the Connections navigator, and select REST Services > Enable RESTful Services to display the following wizard page: 
RESTful Services wizard step: Specify Details. Contains fields for Enable Object, Object Alias, and Authorization Required.

**********************************************************************************************

Enable object: Enable this option (that is, enable REST access for the EMP table).
Object alias: Acceptemp for the object alias.
Authorization required: For simplicity, this tutorial does not require authorisation, so disable this option.

8. On the RESTful Summary page of the wizard, click Finish.
The EMP table is now exposed as a REST HTTP endpoint.

9. Test the REST endpoint. In a web browser enter the following URL:
   http://localhost:8080/ords/ordstest/emp/
   - The ORDSTEST schema has been exposed at the /ordstest/ path.
   - The EMP table has been exposed at the /emp/ path.

A JSON document similar to the following is displayed:
JSON document returned by the URL used to test the REST endpoint.
E.2 Create a RESTful Service from a SQL Query

Oracle REST Data Services provides a REST API (called the Resource Modules API) that enables Oracle SQL Developer to create and edit RESTful service definitions. Access to the Resource Modules API is protected, a user with the correct role must be provisioned, and the created user's credentials must be used when accessing the API from SQL Developer.

To create a RESTful service from a SQL query, follow these steps.

1. In the folder where Oracle REST Data Services was installed, enter the following command at a command prompt:
   
   ```
   java -jar ords.war user test_developer "SQL Developer"
   ```
   
   - You will be prompted to enter a password.
   - This command creates a user named test_developer and grants the user the role named SQL Developer. Only users with the SQL Developer role are permitted to access the resource module's API.
   - The user details are stored in a file named credentials in the ORDS configuration folder. However, it is not recommended to store user credentials in the credentials file in production deployments; instead, users should be provisioned in the host application server.

   The remaining steps create and test the RESTful service.

2. Create RESTful connection. In SQL Developer, select View > REST Data Services > Development.

3. In the REST Development pane, right-click REST Data Services > Connect.

4. In the RESTful Services Connection dialog box, click the + (plus sign) icon to add a connection to the list available for selection.

5. In the New RESTful Services Connection dialog box, enter the necessary information:
Create a RESTful Service from a SQL Query

Getting Started with RESTful Services

New RESTful Services dialog box, containing the options described next in the text.

Connection Name: Any desired name for the connection. Example: ordstest
Username: test_developer
http or https: Select http for simplicity in this tutorial.
Hostname: localhost
Port: 8080
Server Path: /ords
Workspace: ordstest

Click OK, then enter the password for the test_developer user at the prompt.

6. Create the module. Right-click the Modules node in the REST Development view, click New Module, and enter information on the Specify Module page:
RESTful Services wizard Specify Module page, containing the options described next in the text.

- **Module Name**: Any desired name for the connection. Example: `test`
- **URI Prefix**: `/test`
- **Publish - Make this RESTful Service available for use**: Enable (check).
- **Pagination Size**: 7

7. Click **Next**, and enter information on the Specify Template page:
RESTful Services wizard Specify Template page, containing the options described next in the text.

- URI Template: /emp/
- Accept the defaults for the remaining options.

8. Click Next, and enter information on the Specify Handler page:
Create a RESTful Service from a SQL Query

RESTful Services wizard Specify Handler page, containing the options described next in the text.

- **Method**: GET
- **Requires Secure Access**: Disable (uncheck) for this tutorial.
- **Source Type**: Collection Query
- **Pagination Size**: 7

9. Click Next to go to the RESTful Summary page of the wizard, then click **Finish**. The resource module is now created, the next step is to define the query for the GET resource handler.

10. Define the query for the GET resource handler.

   a. Expand the **test** node under the **Modules** node in the REST Development view.
   b. Expand the */emp/ node, right-click the **GET** node, and select **Open**.
   c. In the SQL Worksheet that opens for **GET /emp/**, enter the following SQL query:
      ```sql
      select * from emp
      ```
   d. Right-click on the test node under the 'Modules' node in the 'REST Development' view
   e. Click 'Upload...'. A confirmation dialog will appear confirming the module has been uploaded.

11. Test the RESTful service. In a web browser enter the following URL:
http://localhost:8080/ords/ordtest/test/emp/

- The ORDSTEST schema has been exposed at the /ordstest/ path.
- The EMP table has been exposed at the /test/emp/ path.

A JSON document similar to the following is displayed:
Create a RESTful Service from a SQL Query

JSON document returned by the URL used to test the RESTful service.

E-12  Oracle REST Data Services Installation, Configuration, and Development Guide
E.3 Protect Resources

Up to this point the tutorial has deliberately disabled security on the RESTful endpoints you created, because it is easier to test them without security. In this topic you protect the /test/emp/ service, requiring users to authenticate before accessing the service.

Controlling access to protected resources is done by defining privileges. Privileges restrict access to only users having at least one of a set of specified roles. A privilege is then associated with one or more resource modules: before those resource modules can be accessed, the user must be authenticated and then authorized to ensure that the user has one of the required roles.

To protect resources, follow these steps.

1. Create a privilege. In SQL Developer, right-click the Privileges node in the REST Development view and select New Privileges to display the Edit Privilege dialog box:

![Edit Privilege dialog box: Contains options mentioned in text that follows.](image)

Edit Privilege dialog box: Contains options mentioned in text that follows.

Name: test
Title: Example Privilege
Description: Demonstrate controlling access with Privileges
Protected Modules: Ensure that the list includes the test module. Use the arrow button to move it if necessary.

Click Apply.

2. Right click the test privilege and click Upload.

A dialog box confirms that the privilege has been uploaded.

You have now created a privilege that protects the test module. However, you have not restricted the privilege to any particular role; this will just require that the user be authenticated before accessing the test module (the next step).

3. Test the RESTful service. In a web browser enter the following URL:

   http://localhost:8080/ords/ordstest/test/emp/

4. Click the link to sign in, and enter the test_developer credentials.

   The contents of the JSON document are displayed.

E.4 Register an OAuth Client Application

This topic explains how to register your applications (called "third-party" applications here) to access a REST API.

OAuth 2.0 is a standard Internet protocol that provides a means for HTTP servers providing REST APIs to give limited access to third party applications on behalf of an end user.

- The author of the third-party application must register the application to gain client credentials.
- Using the client credentials the third party application starts a web flow that prompts the end-user to approve access.

So, before a third party application can access a REST API, it must be registered and the user must approve access. And before the application can be registered, it must be assigned a user identity that enables the user to register applications. Users possessing the SQL Developer role (such as the test_developer user created in Create a RESTful Service from a SQL Query) are permitted to register OAuth clients.

Tip: In a real application, you may want to provision specific users that can register OAuth clients; these users should be granted the OAuth Client Developer role.

This topic outlines how to complete these actions. It is not a full-featured demonstration of how to create and integrate a third party application; it just outlines the concepts involved.

1. Register the client application.

   a. In a web browser enter the following URL:

      http://localhost:8080/ords/ordstest/oauth/admin/clients/

   b. At the prompt, click the link to sign in and enter the credentials for the test_developer user.

   c. Click New Client and enter the following information:

      Name: Test Client

      Description: An example OAuth Client
Redirect URI: http://example.org/redirect
Support e-mail: info@example.org
Support URI: http://example.org/support
Required Privileges: Example Privilege

d. Click Create.

The client registration is created, and the Authorization URI for the client is displayed. You have created a client that will use the Implicit Grant authorization flow (explained at https://tools.ietf.org/html/rfc6749#section-4.2).

Note the Client Identifier assigned to the client and the Authorization URI value. These values are used to start the authorization flow (next major step).

2. Approve the client application.

In a real third-party client application, the client will initiate the approval flow by directing a web browser to the Authorization URI. The end user will be prompted to sign in and approve access to the client application. The browser will be redirected back to the client's registered Redirect URI with a URI fragment containing the access_token for the approval. To simulate this process:

a. In a web browser, enter the Authorization URI that you noted in the previous step. The URL should look like the following (though you should not copy and paste in this example value):

   http://localhost:8080/ords/ordstest/oauth/auth?response_type=token&client_id=5B77A34A266EFB0056BE3497ED7099.&state=d5b7944-d27d-8e2c-4d5c-fb80e1114490&_auth_=force

   The client_id value must be the value of the client identifier assigned to the application. Be sure you are using the correct client_id value. Do not use the value in the preceding example; replace it with the client identifier assigned to your application.

   The state value should be a unique, unguessable value that the client remembers, and can use later to confirm that the redirect received from Oracle REST Data Services is in response to this authorisation request. This value is used to prevent Cross Site Request Forgery attacks; it is very important, cannot be omitted, and must not be guessable or discoverable by an attacker.

b. At the prompt, click the link to sign in and enter the credentials for the test_developer user.

c. Review the access being requested, and click Approve.

   The browser is redirected to a URL similar to the following:

   http://example.org/redirect#token_type=bearer&access_token=-i_Ows8j7JYu0p07jOPMEA..&expires_in=3600

   When registering the OAuth client, you specified http://example.org/redirect as the Redirect URI. On completion of the approval request, the browser is redirected to this registered redirect URI. Appended to the URI is the information about the access token that was generated for the approval.

   In a real application, the third party application would respond to the redirect to the redirect URI by caching the access token, redirecting to another page to show the user that they are now authorized to access the REST API, and
including the access token in every subsequent request to the REST API. However, in this tutorial you just make note of the access token value and manually create a HTTP request with the access token included, as explained in the next major step.

The value of the access token (which in the preceding example is -i_0w8j7JYu0p07jOFMEA..) will change on every approval.

Note that the access token expires. In the preceding example it expires after 3600 seconds (expires_in=3600), that is, one hour.

3. Issue an authorized request.

After an access token has been acquired, the client application must remember the access token and include it with every request to the protected resource. The access token must be included in the HTTP Authorization request header (explained at http://tools.ietf.org/html/rfc2616#section-14.8) as in the following example:

```text
Host: localhost:8080
GET /ords/ordstest/test/emp/
Authorization: Bearer -i_0w8j7JYu0p07jOFMEA..
```

To emulate creating a valid HTTP request, use the cURL command line tool (if necessary, install cURL using http://curl.haxx.se/). In a real application this request would be performed by the client making an HTTP request, such as an XMLHttpRequest. For example:

```bash
curl -i -H'Authorization: Bearer -i_0w8j7JYu0p07jOFMEA..' 
http://localhost:8080/ords/ordstest/test/emp/
```

However, in this example replace -i_0w8j7JYu0p07jOFMEA.. with the access token value that you previously noted.

Output similar to the following JSON document should be displayed:

```json
HTTP/1.1 200 OK
Content-Type: application/json
Transfer-Encoding: chunked

{
   "items": [
      {
         "empno":7369, "ename":"SMITH", "job":"CLERK", "mgr":7902, "hiredate":"1980-12-17T0 00:00:00Z", "sal":800, "comm":null, "deptno":20},

      {
         "empno":7499, "ename":"ALLEN", "job":"SALESMAN", "mgr":7698, "hiredate":"1981-02-2 0700:00:00Z", "sal":1600, "comm":300, "deptno":30},

      {
         "empno":7521, "ename":"WARD", "job":"SALESMAN", "mgr":7698, "hiredate":"1981-02-22 00:00:00Z", "sal":1250, "comm":500, "deptno":30},

      {
         "empno":7566, "ename":"JONES", "job":"MANAGER", "mgr":7839, "hiredate":"1981-04-01 T23:00:00Z", "sal":2975, "comm":null, "deptno":20},

      {
         "empno":7654, "ename":"MARTIN", "job":"SALESMAN", "mgr":7698, "hiredate":"1981-09-27T23:00:00Z", "sal":1250, "comm":1400, "deptno":30},

      {
         "empno":7698, "ename":"BLAKE", "job":"MANAGER", "mgr":7839, "hiredate":"1981-04-30 T23:00:00Z", "sal":2850, "comm":null, "deptno":30},

      {
         "empno":7782, "ename":"CLARK", "job":"MANAGER", "mgr":7839, "hiredate":"1981-06-08
```

```
However, if the Authorization header is omitted, then the status 401 Unauthorized is returned instead.
A

access token
  acquiring, C-12
  securing, C-18
  using, C-13
Apache Tomcat, 1-2
  about, 1-18
  configuring Oracle REST Data Services images, 1-18
  deploying to, 1-18
downloading, 1-18
authentication
  against WebLogic and GlassFish user repositories, 3-46

B

browser origins, C-14

C

command-line interface, 1-3
cfgdir command, A-1
  locating configuration files, A-1
  locating configuration folder, A-1
configuration file editable parameters
  apex.docTable, A-3
  apex.excel2collection, A-3
  apex.excel2collection.name, A-3
  apex.excel2collection.onecollection, A-3
  apex.excel2collection.useSheetName, A-3
  cache.caching, A-4
  cache.directory, A-4
  cache.duration, A-4
  cache.expiration, A-4
  cache.maxEntries, A-4
  cache.monitorInterval, A-4
  cache.procedureNameList, A-4
  cache.type, A-4
db.connectionType, A-4
db.customURL, A-5
db.hostname, A-5
db.password, A-5
db.port, A-5
db.servicename, A-5
db.sid, A-5
db.tnsAliasName, A-5
db.tnsDirectory, A-5
db.username, A-5
debug.debugger, A-5
debug.printDebugToScreen, A-5
error.keepErrorMessages, A-5
error.maxEntries, A-6
icap.port, A-6
icap.server, A-6
dbc.axRows, A-6
dbc.DriverType, A-6
dbc.InactivityTimeout, A-6
dbc.InitialLimit, A-6
dbc.MaxConnectionReuseCount, A-6
dbc.MaxLimit, A-6
dbc.MaxStatementsLimit, A-6
dbc.MinLimit, A-6
log.logging, A-7
log.maxEntries, A-7
log.procedure, A-7
misc.defaultPage, A-7
procedure.postProcess, A-7
procedure.preProcess, A-7
security.disableDefaultExclusionList, A-7
security.exclusionList, A-7
security.inclusionList, A-8
security.maxEntries, A-8
security.requestValidationFunction, A-8
security.verifySSL, A-8
soda.defaultLimit, A-8
soda.maxLimit, A-8
statementTimeout, A-7
configuration files, A-1
  format of, A-2
  locating using cfgdir command, A-1
configuration folder
  setting location, A-1
  structure of, A-1
CORS (Cross Origin Resource Sharing), C-14
CREATE_APPROVAL_PRIVILEGE procedure, 6-2, 6-3, 6-4, 6-5, 6-6, 6-7
CREATE_ROLE procedure, 5-2
CREATE_SERVICE procedure (deprecated), 5-3
Cross Origin Resource Sharing (CORS), C-14
Cross Site Request Forgery (CSRF) attacks, C-12
DEFINE_PARAMETER, 5-7
DEFINE_PRIVILEGE, 5-8
DEFINE_SERVICE, 5-10
DEFINE_TEMPLATE, 5-13
DELETE_MODULE, 5-14
DELETE_PRIVILEGE, 5-15
DELETE_ROLE, 5-16
DROP_REST_FOR_SCHEMA, 5-17
ENABLE_OBJECT, 5-18
ENABLE_SCHEMA, 5-19
PUBLISH_MODULE, 5-20
RENAME_MODULE, 5-21
RENAME_PRIVILEGE, 5-22
RENAME_ROLE, 5-23
SET_MODULE_ORIGINS_ALLOWED, 5-24
SET_URL_MAPPING, 5-25

P
PUBLISH_MODULE procedure, 5-20

R
RENAME_MODULE procedure, 5-21
RENAME_PRIVILEGE procedure, 5-22
RENAME_ROLE procedure, 5-23
resource handler, 3-3
resource module, 3-3
resource template, 3-3
RESTful services
about, 3-2
accessing, 3-3
accessing from third-party application, C-10
configuring for cross-origin resource
sharing, C-14
developing, 2-4
going started with, 3-2
image gallery example, C-1
integrating with existing group/role
models, 3-49
sample services, 3-4
securing, C-8
terminology, 3-2
user roles, 3-44
using cURL, 3-9
role-mapping.xml file, 3-49
route pattern, 3-3

S
Same Origin Policy, C-14
SET_MODULE_ORIGINS_ALLOWED
procedure, 5-24
SET_URL_MAPPNG procedure, 5-25
SQL Developer Oracle REST Data Services
Administration, 1-9
standalone mode
starting, 1-11
stopping the server, 1-12
standalone mode, running in, 1-11
structure of configuration folder, A-1

T
troubleshooting, B-1
enabling debug tracing, B-1
enabling detailed request error messages, B-1

U
upsert operation, 3-18
URI pattern, 3-3
URI template, 3-3
url-mapping.xml
file format, A-2
request rules routing, A-2
user roles for RESTful services, 3-44

supported Java EE application servers, 1-2
system requirements, 1-2