Oracle® Fusion Middleware

Developer's Guide for Oracle API Catalog

12*c* Release 1 (12.1.3)

E55981-03

August 2015

Describes how to use the Oracle API Catalog Console and how to install and use the Oracle Enterprise Repository Plug-in for Oracle JDeveloper. This guide also describes how to use the Repository Extensibility Framework.

ORACLE

Oracle Fusion Middleware Developer's Guide for Oracle API Catalog, 12*c* Release 1 (12.1.3)

E55981-03

Contents

Preface

Oracle Fusion Middleware Developer's Guide for Oracle API Catalog describes how to use Oracle API Catalog console and how to connect Oracle JDeveloper to interact with Oracle API Catalog. This guide also describes how to use the Repository Extensibility Framework.

Audience

This document is intended for all Oracle API Catalog users who want to use the Oracle API Catalog console or who want to configure Oracle JDeveloper to easily consume files from Oracle API Catalog. This document is also intended for all Oracle API Catalog users who want to use REX and the REX APIs.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

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 or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information, see the following documents in the Oracle API Catalog 12*c* documentation set:

- *Oracle Fusion Middleware Concepts Guide for Oracle API Catalog*
- *Oracle Fusion Middleware Installation Guide for Oracle API Catalog*
- *Oracle Fusion Middleware Administrator's Guide for Oracle API Catalog*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Part I

Introducing Oracle API Catalog Developer Features

This part describes the basic use case for developers using the Oracle API Catalog Web Interface and Oracle Enterprise Repository JDeveloper Plug-in.

This part contains the following chapter:

1

Getting Started with Oracle API Catalog

This chapter describes the basic workflow for a developer using Oracle API Catalog

This chapter includes the following sections:

1.1 Overview

The Oracle API Catalog (OAC) option provides a streamlined subset of Oracle Enterprise Repository features that allow organizations to easily build a catalog of their APIs, providing visibility for these APIs to internal application developers.

See the *Oracle Fusion Middleware Concepts Guide for Oracle API Catalog* for a description of OAC concepts.

1.2 Discovering APIs

The curator populates OAC with APIs and their associated metadata and publishes the APIs for developers to discover and use. See the *Oracle Fusion Middleware Administrator's Guide for Oracle API Catalog* for more information about populating OAC with APIs.

Developers can discover and use the APIs published to OAC using the OAC web interface. Developers can also use this interface to manage a list of APIs of interest and provide ratings and feedback about these APIs.

Chapter 2, "Getting Started with the Oracle API Catalog Console" discusses using the OAC web interface.

1.3 Using the JDeveloper Plug-in

Developers can also use the Oracle Enterprise Repository JDeveloper plug-in to connect with OAC. From the plug-in, developers can browse APIs collected in OAC and consume APIs from OAC.

Chapter 3, "Configuring Oracle JDeveloper to Support Integration with Oracle API Catalog" discusses installing the Oracle Enterprise Repository JDeveloper plug-in.

Chapter 4, "Using Oracle JDeveloper to Interact with Oracle API Catalog" discusses searching for APIs and consuming APIs.

1.4 Introducing the REX API

This manual also covers the REX API, which allows programmatic access to OAC. The REX API can be used to automate some tasks in OAC and would most likely be used to automate tasks related to the production and consumption of APIs.

Chapter 5, "Using the Repository Extensibility Framework" discusses using the REX Framework with Oracle API Catalog.

Part II

Using the Oracle API Catalog Console

This part describes using the Oracle API Catalog console to find, use, and evaluate assets.

This part contains the following chapter:

- Chapter 2, "Getting Started with the Oracle API Catalog Console"

2

Getting Started with the Oracle API Catalog Console

This chapter describes how to use the Oracle API Catalog console.

This chapter includes the following sections:

- Getting Started
- User Role Descriptions
- Exploring the Asset Lifecycle

2.1 Getting Started

This section describes the following processes:

- Section 2.1.1, "Logging in to Oracle API Catalog"
- Section 2.1.2, "The Default View"
- Section 2.1.3, "Logging Out of Oracle API Catalog"

2.1.1 Logging in to Oracle API Catalog

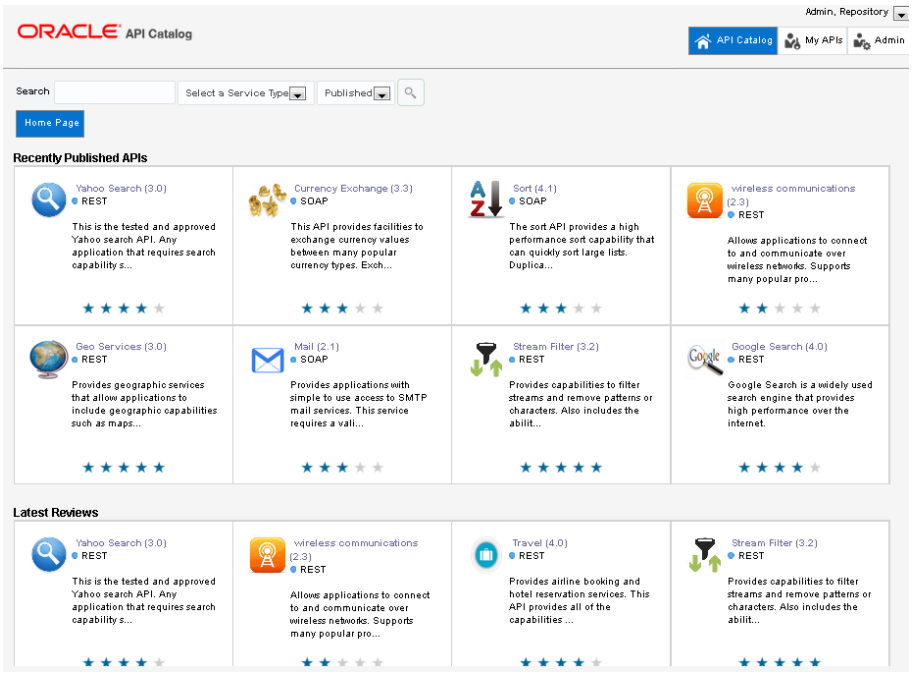
To log in to Oracle API Catalog, perform the following steps:

1. Using a browser, open Oracle API Catalog.
2. On the login screen, enter the appropriate information in the Username and Password boxes.
3. Click **Sign In**.

2.1.2 The Default View

After you have logged in, you will see the Oracle API Catalog home page, as shown in Figure 2–1. Note that the APIs in the image are for illustrative purposes only; OAC does not include example APIs.

Figure 2–1 Oracle API Catalog Home Page



This image displays the Oracle API Catalog home page.

****

The Oracle API Catalog Menu Bar

The Oracle API Catalog menu bar runs across the top of the console, and enables you to navigate to the various tools and features within Oracle API Catalog.

The elements that appear in the menu bar are determined by the role of the user who is currently signed in. For example, the list with published status options in the search region and the Admin button do not appear when a user with the developer role is signed in. See Section 2.2, "User Role Descriptions" for more information about user roles in Oracle API Catalog.

- **Search**

  The search region is the primary method of discovering APIs in API Catalog.

  Use the search region to find assets using criteria of your choice. See Section 2.3.1.1, "Performing Standard Searches" for more information about searches.

- **API Catalog Home**

  Click the API Catalog Home button to navigate to the API Catalog home page. The API Catalog home page displays the API Search, in addition to recently published and recently reviewed APIs.

- **My APIs**

  Click the My APIs button to navigate to the My APIs page. The My APIs page collects all of the APIs that you have declared interest in, providing quick access to the detail and review submission pages for these APIs.

  See Section 2.3.3, "Managing My APIs" for more information about the My APIs page.

- **Admin**

  Click the Admin button to navigate to the Admin page. From the Admin page, users with the appropriate roles can configure the infrastructure of API Catalog. For more information on these tasks, see the *Oracle Fusion Middleware Administrator's Guide for Oracle API Catalog*.

  Note: The Admin page is available only to users with the admin role.

- **User Menu**

  Click your name to expand the User Menu. From this menu. you can access Help, change your password, or sign out of Oracle API Catalog.

2.1.3 Logging Out of Oracle API Catalog

To log out of Oracle API Catalog:

1. From the Oracle API Catalog menu bar, click the menu icon to the right of the user name to expand the user menu.
2. Select **Sign Out**.

2.2 User Role Descriptions

Oracle API Catalog users are defined by roles. Each role has certain permissions that enable users to fulfill specific tasks.

The default roles shipped with Oracle API Catalog are described below.

- **developer**: Users with the developer role have the ability to search OAC for registered APIs from the OAC console or using the Oracle Enterprise Repository JDeveloper plug-in. The developer can examine the API metadata to better understand the API. The developer also has the ability to declare interest in the API and submit ratings and reviews for an API.

  This guide describes the tasks performed by users with the developer role.

- **curator**: In addition to the capabilities available to the developer role, users with the curator role can run the harvester to create new API assets in OAC. After API assets have been created, curators edit them to update their metadata. The curator also has the ability to publish an API, which makes the API available for discovery by developers.

  For more information about the lifecycle tasks available to users with the curator role, see the "Introducing Asset Lifecycle Administration Tasks" section in the *Oracle Fusion Middleware Administrator's Guide for Oracle API Catalog*.

- **admin**: In addition to the capabilities available to the curator and the developer roles, users with the admin role have access to the **Admin** page in OAC. From this page users with this role can administer the infrastructure of OAC by editing system settings, creating new users, creating new departments, managing sessions, and using the import/export tool. Users with this role can also configure the security features included with OAC.

  For more information about infrastructure administration tasks available to users with the admin role, see the "Introducing Infrastructure Administration Tasks" section in the *Oracle Fusion Middleware Administrator's Guide for Oracle API Catalog*.

For more information about the security configuration tasks available to users with the admin role, see the "Introducing Security Administration Tasks" section in the *Oracle Fusion Middleware Administrator's Guide for Oracle API Catalog*.

2.3 Exploring the Asset Lifecycle

This section describes how you can work with assets in Oracle API Catalog. This section includes the following topics:

- Section 2.3.1, "Finding Assets"
- Section 2.3.2, "Evaluating Assets"
-
- Section 2.3.3, "Managing My APIs"
- Section 2.3.4, "Consuming URLs from an API's Detail Page"
- Section 2.3.5, "Reviewing Assets"
- Section 2.3.6, "Exporting Asset Details"

Note: The only asset type available for Oracle API Catalog is the API Asset type.

2.3.1 Finding Assets

You can find assets in Oracle API Catalog using various methods. This section includes some of the methods to find assets in Oracle API Catalog:

- Section 2.3.1.1, "Performing Standard Searches"

Note: The only asset type available for Oracle API Catalog is the API Asset type.

2.3.1.1 Performing Standard Searches

Perform searches from the Search region in the Oracle API Catalog menu bar.

1. Enter a keyword into the **Search** field.
2. **(Optional)** Select a service type (All Services, REST, or SOAP) from the list.
3. **(Optional)** Select a published status (Published, Draft, or All APIs) from the list.

Note: You must have the curator or admin roles to view the list with published status options in the search region.

---

4. Click the **Search** button. The Search Results page displays the results.

2.3.1.1.1 Searching For a Specific Field

You can search specific fields, such as Name and Description.

Note: The `cmee.search.specific.field` system setting must be enabled to search specific fields. See "System Settings Overview" in the *Oracle Fusion Middleware Administrator's Guide for Oracle API Catalog* for more information about system settings.

- Use the following syntax to perform a search on myAsset in the name field only:

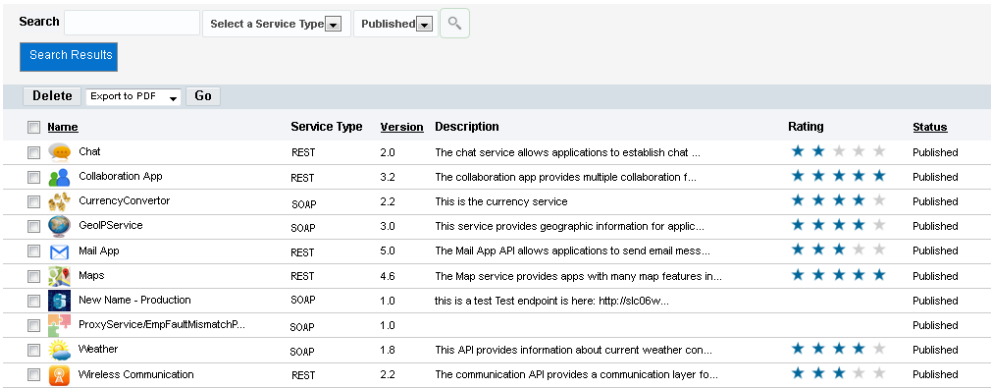
  `name:myAsset`

- Use the following syntax to perform a search on myAsset in the description field only:

  `description:myAsset or desc:myAsset`

Search results appear in the **Search Results** tab, as displayed in Figure 2–2.

Figure 2–2 Oracle API Catalog Search Results Tab



The Oracle API Catalog Search Results tab is displayed. The Search field is empty. Select a Service Type and Published are displayed in the search lists. Export to PDF is displayed in the Export To list. Ten API results are displayed that match the current search criteria.

****

From the Search Results tab, you can export the details of the results to a zip, PDF, or Microsoft Excel file. See Section 2.3.6, "Exporting Asset Details" for more information.

Users with the admin role can also delete assets from OAC from the search results page. See the "Editing Metadata of API Assets in Oracle API Catalog" chapter in the *Oracle Fusion Middleware Administrator's Guide for Oracle API Catalog* for more information.

2.3.1.2 Finding APIs on the API Catalog Home Page

The API Catalog home page features assets that have recently been published or reviewed. This allows you to quickly access the most recently changed APIs that you may be interested in.

To find recently published APIs:

1. Click the **API Catalog Home** button to navigate to the API Catalog home page.
2. Review the APIs displayed in the Recently Published APIs section. The APIs that a curator have most recently published are displayed here.

To find recently reviewed APIs:

1. Click the **API Catalog Home** button to navigate to the API Catalog home page.
2. Review the APIs displayed in the Latest Reviews section. The APIs that have been reviewed most recently are displayed here.

2.3.2 Evaluating Assets

The API Detail can be viewed in tabular or frame format. The API Detail page provides a wide variety of information on the use, functionality, and history of an API. Typically, asset metadata is provided and reviewed by the curator before presentation through Oracle Enterprise Repository.

After performing a search, click any asset listed in the search results to open its API Detail in a new tab.

Search results on the Asset Search in the Web Console are now limited by default, to improve initial search performance. The number of results is configurable on the Admin | System Properties screen: "`cmee.search.assets.maxresults`". If a search exceeds the maximum number of results, the search results screen shows a result count like "Results (1000 of 2345)", and a link "Show All" that brings back all of the results.

Note: The only asset type available for Oracle API Catalog is the API Asset type.

2.3.3 Managing My APIs

My APIs collects the APIs that you are interested in. Every API that you've added to your My APIs list is displayed on your My APIs page. Each user has their own My APIs page, so APIs that another user has expressed interest in but you have not expressed interest in will not be displayed. Once an API has been added to My APIs, you can add reviews and ratings viewable by other users.

This section describes the following tasks:

- Section 2.3.3.1, "Adding an API to My APIs"
- Section 2.3.3.2, "Removing an API from My APIs"
- Section 2.3.3.3, "Viewing an API's Usage History"

2.3.3.1 Adding an API to My APIs

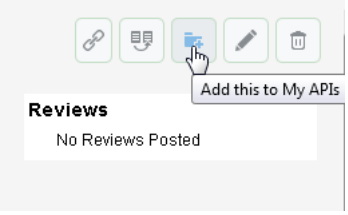
You should add an API to your My APIs list if you are using it or are interested in using it.

Note: You should always add APIs that you use to My APIs. This is how usage metrics are determined in Oracle API Catalog.

To add an API to My APIs:

1. Find an API that you are interested in by performing a search, as described in Section 2.3.1.1, "Performing Standard Searches".
2. Click on an API on the Search Results page to open that API's detail page.
3. After reviewing the API's metadata, click the **Add to My APIs** icon, as shown in Section 2–3, "Add to My APIs Icon". This API now appears on the My APIs page.

Figure 2–3 Add to My APIs Icon



This graphic displays a portion of an asset page in Oracle API Catalog. The figure displays the Link to Detail Page icon, View Details of this Item in Tabs, the Add to My APIs icon (marked by the mouse pointer), the Edit Details, and the Delete icon are displayed.

****

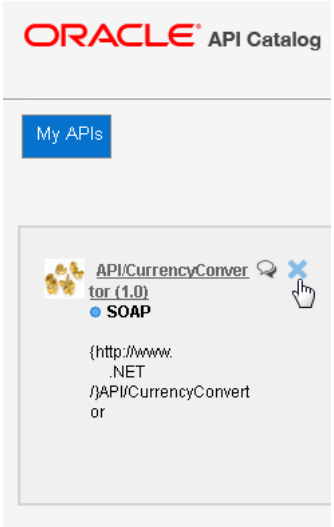
> **Note:** If the API has already been added to My APIs, the Submit a Review icon replaces the Add to My APIs icon.

2.3.3.2 Removing an API from My APIs

To remove an API from the My APIs:

1. Click the **My APIs** button to display all of the APIs that have been added.
2. Click the **Remove** icon for each API that you want to remove, as shown in Figure 2–4.

Figure 2–4 Remove API Icon



This image displays an API on the My APIs page. The Submit a Review icon and Remove API icons are displayed. The mouse pointer is hovering over the Remove API icon.

****

2.3.3.3 Viewing an API's Usage History

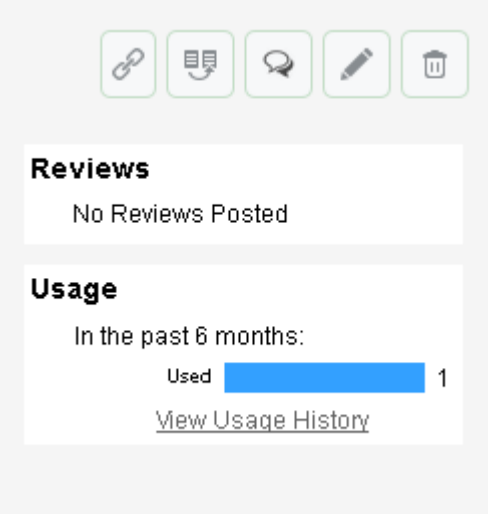
Viewing an API's history gives you insight into how the API is being used. Oracle API Catalog tracks how many users have added an API to their My APIs page.

> **Note:** An API must first be added to a user's My APIs before that user can track its usage. See Section 2.3.3.1, "Adding an API to My APIs" for more information.

To view the history of an API:

1. Click on the **My APIs** button to open My APIs.
2. Find the API you want to review on the My APIs page. Click on its title to open the API's detail page.
3. On the API's detail page, click View Usage History, as shown in Figure 2–5.

Figure 2–5 Usage Region on an API Detail Page



This image displays a portion of an API's detail page. The Link to Detail Page, Toggle Tabbed View, Submit a Review, Edit Asset Details, and Delete icons are displayed. Below these icons, the Reviews region is displayed. At the bottom of the image, the Usage region is displayed. This region indicated that the API is added to one user's My APIs. The View Usage History link can be clicked to view a more detailed usage history.

****

The Usage dialog opens. The Used graphs display how many users have added this API to their My API page over the given time spans. The table at the bottom of the dialog display the users that currently have this API added to My APIs.

2.3.4 Consuming URLs from an API's Detail Page

An API's detail page includes information to help you incorporate the API into your application. Using the OAC console, you can find an endpoint URL that you can use to interact with the API. You can also find the URL of a WSDL or WADL file that you can import into your IDE.

2.3.4.1 Using the Endpoint URL of an API

To find the endpoint for an API that can be consumed into your IDE:

1. Find the API for which you want to view the endpoint URL in your **My APIs** list. Open its detail page.
2. The Endpoint URL is displayed in the **Technical Information** region.
3. You can interact with the API through this URL by copying and pasting it into your code or your IDE.

2.3.4.2 Using the WSDL or WADL of an API

To view the WSDL or WADL file associated with the API:

1. Find the API for which you want to view the WSDL or WADL file in your **My APIs** list. Open its detail page.
2. The region in which the URL is displayed differs depending on if you are viewing a SOAP service or REST service:
   - For a SOAP service, the **File Location URL** is displayed in the **WSDL Summary** region.
   - For a REST service, the **WADL URL** is displayed in the **Technical Information** region.
3. Click the URL to view the file's contents. You can also right-click the URL, save the file, and import it into your application in your IDE.

2.3.5 Reviewing Assets

Individuals who have used an asset can provide a written review and a rating for the asset. These reviews are useful to other users, as well as to those who manage the assets. It provides an opportunity for users to share their experiences with others in the community.

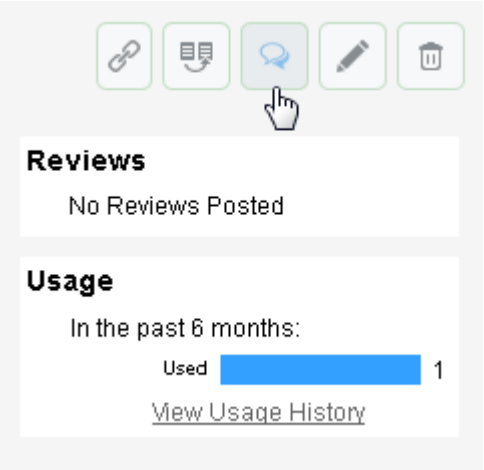
> **Note:** The only asset type available for Oracle API Catalog is the API Asset type.

Follow these steps to rate an asset and to submit a review.

> **Note:** You can only review assets that have been added to My APIs.

1. Find the API you want to review on the My APIs page. Click on its title to open the API's detail page.
2. Click **Submit a Review** icon, as shown in Figure 2–6. The Submit a Review dialog is displayed, as shown in Figure 2–7.

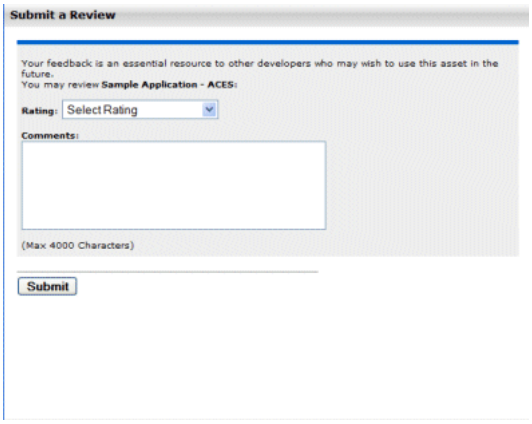
Figure 2–6 Submit a Review Icon



This graphic displays a portion of an asset page in Oracle API Catalog. The figure displays the Link to Detail Page icon, the View Details of this Item in Tabs icon, the Submit a Review icon (marked by the mouse pointer), the Edit Details icon, and the Delete icon are displayed. The Reviews panel is displayed at the bottom of the image. The Reviews panel shows a rating of two stars for this asset. The View All Reviews link appears below the asset's rating.

****

Figure 2–7 Submit a Review



3. Select the appropriate rating from the **Rating** list.
4. Enter the appropriate review information in the **Comments** text box.
5. Click **Submit**. A confirmation message appears.
6. Close the window, or click **click here** to display the review.

2.3.6 Exporting Asset Details

The details of an API can be exported in any one of the following forms:

- Section 2.3.6.1, "Exporting Detail to Excel"
- Section 2.3.6.2, "Exporting Detail to PDF"

- Section 2.3.6.3, "Exporting Asset Search Results to ZIP"

2.3.6.1 Exporting Detail to Excel

You can export API information to an Excel spreadsheet. This may be helpful if you want to import an API's information into another tool.

You can export details to an Excel file using the following methods:

- Section 2.3.6.1.1, "Export an Excel File from Search Results"
- Section 2.3.6.1.2, "Export an Excel file from the Detail Page"

> **Note:** The export will fail if a field containing a hyperlink (for instance, the Endpoint URI or WADL URL fields) contains more than 128 characters.

2.3.6.1.1 Export an Excel File from Search Results

This procedure is performed on the Oracle API Catalog home page.

1. Perform a search by entering keywords in the **Search** field. Select a service type and published status from the lists, as necessary, to refine the search.
2. Select one or more APIs from the search results by placing a check in the check box that appears to the left of APIs listed in the Search Results page.
3. Select **Export to Excel** in the list at the top of the pane.

> **Note:** The Export to Excel option is displayed, only when the `cmee.asset.registry.export.excel` property is enabled. This is enabled by default, but if you are migrating, then you might face an issue if this property is not enabled.

4. Click **Go**. Export progress is indicated in the progress window, with the following options:
   - Click **Terminate Job** to end the export.
   - Click **View Audit Log** to open a log of the export process.
5. When the export is complete, click **View Excel Spreadsheet** in the progress window. The spreadsheet opens in Microsoft Excel.

   APIs are listed horizontally; API Detail elements are indicated by the light-green background.

2.3.6.1.2 Export an Excel file from the Detail Page

This procedure is performed on the Oracle API Catalog home page.

1. Locate the appropriate API.
2. Click the **Excel** icon in the detail page. Export progress is indicated in the progress window.
3. When the export is complete, click **View Excel Spreadsheet** in the progress window to open the spreadsheet in Microsoft Excel.

2.3.6.2 Exporting Detail to PDF

This procedure is performed on the Oracle API Catalog home page.

2.3.6.2.1 Exporting from the Detail Page

You can export the asset details to PDF from the Detail page.

1. Use Search or other means to locate the appropriate API.
2. Click the **PDF** icon in the API detail.
3. If prompted to do so, select a printing template from the list.

   If the Adobe Acrobat plug-in is installed, the browser opens the PDF. Otherwise, you are prompted to save or open the file.

2.3.6.2.2 Exporting from the Search Results List

1. Perform an API search by entering search terms in the Search field.

   (Optional) Select a service type or published status from the lists as necessary to refine the search.

   > **Note:** You must have the curator or admin roles to view the list with published status options in the search region.

2. Select one or more APIs from the search results by placing a check in the check box that appears to the left of APIs listed in the Search Results.
3. Select **Export to PDF** from the list above the list of APIs.
4. Click **Go**.
5. If the Adobe Acrobat plug-in is installed, the browser opens the PDF. Otherwise, you are prompted to save or open the file.

2.3.6.3 Exporting Asset Search Results to ZIP

When enabled, the Export to ZIP feature allows the export of asset information to a ZIP file, through a menu selection on the asset search results screen.

- The Export to ZIP feature is available only to users with Administrator permissions.
- Import/Export must be enabled. For more information, see the "System Settings Overview" chapter in the *Oracle Fusion Middleware Administrator's Guide for Oracle API Catalog*.

This procedure is performed on the Oracle API Catalog home page.

1. Perform an asset search by entering keywords in the **Search** field. Select a service type or published status as necessary to refine the search.

   > **Note:** You must have the curator or admin roles to view the list with published status options in the search region.

2. Use the check boxes in the search results list to select the APIs to be exported to ZIP.
3. Select **Export to ZIP** from the list.

4. Click **Go**. The Export to ZIP progress dialog is displayed.
5. When the export is complete, click **Download** to download the exported zip file. The File Download dialog is displayed.
6. Select **Open**, **Save**, or **Cancel**, as appropriate.

Part III

Using JDeveloper with Oracle API Catalog

This part describes configuring JDeveloper to connect to Oracle API Catalog and using JDeveloper to interact with Oracle API Catalog.

This part contains the following chapters:

3

Configuring Oracle JDeveloper to Support Integration with Oracle API Catalog

This chapter describes how to configure Oracle JDeveloper to integrate with Oracle API Catalog. First, you must download the Oracle Enterprise Repository plug-in for JDeveloper. Next, you can create a connection to an Oracle API Catalog server. After installing the plug-in and creating a repository connection, see Chapter 4, "Using Oracle JDeveloper to Interact with Oracle API Catalog" for the next steps.

This chapter includes the following sections:

3.1 Install the Oracle Enterprise Repository JDeveloper 12*c* Plug-in

To install the Oracle Enterprise Repository plug-in for Oracle JDeveloper 12*c*:

1. Install Oracle JDeveloper on your local computer. Starting in 12*c*, you must use the Oracle SOA Suite Quick Start distribution to obtain a version of JDeveloper pre-configured for Oracle SOA Suite. See "Installing Oracle SOA Suite Quick Start for Developers" in *Installing SOA Suite and Business Process Management Suite Quick Start for Developers* for more information.

   > **Note:** Ensure that you are using the Oracle SOA Suite Quick Start for Developers distribution. The Oracle BPM Suite Quick Start for Developers distribution includes a version of JDeveloper that will not work with the plug-in and is not supported for use with Oracle API Catalog.

2. Acquire the file for patch 19721053 (`p19721053_121300_Generic.zip`) from My Oracle Support.
3. Ensure that JDeveloper is closed before continuing.
4. Follow the instructions in the patch's `README.txt` file.
5. Restart JDeveloper.

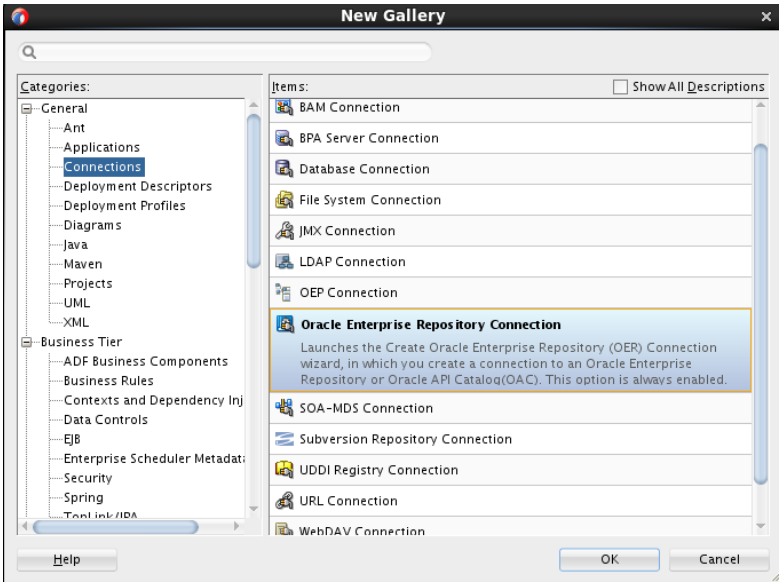
3.2 Create a New Repository Connection

After you have installed the plug-in, you can create a connection to a repository server.

To create a new repository connection:

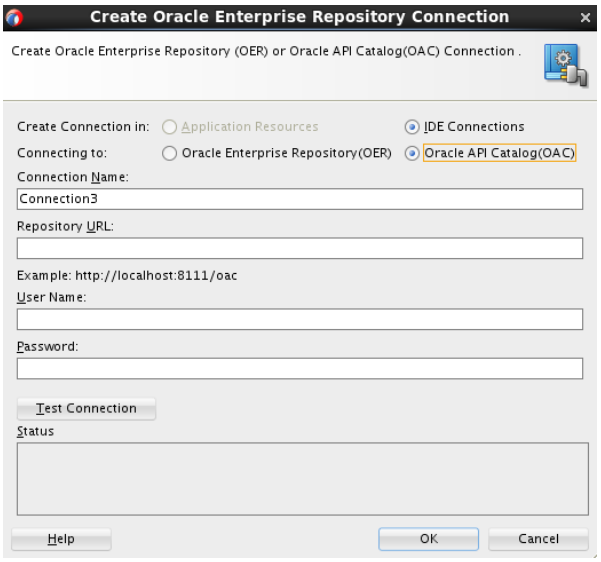
1. Click **File**, **New**. The New Gallery dialog is displayed.
2. Select **General**, **Connections**, and then select **Oracle Enterprise Repository Connection**, as shown in Figure 3–1.

Figure 3–1 New Gallery Dialog



3. Click **OK**. The Create Oracle Enterprise Repository Connection dialog is displayed, as shown in Figure 3–2.

Figure 3–2 Create Oracle Enterprise Repository Connection Dialog



This figure illustrates the Create Oracle Enterprise Repository Connection dialog. This dialog is described in the surrounding text.

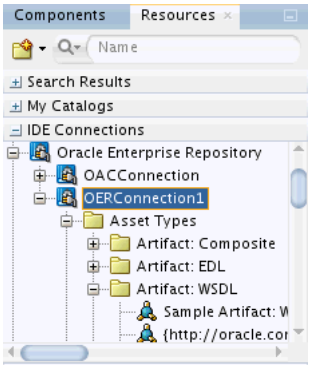
****

4. Enter the following information:

- Connecting To: Select **Oracle Enterprise Repository** if you are connecting to OER, or select **Oracle API Catalog** if you are connecting to OAC.
- Connection Name: A descriptive name for the Oracle API Catalog connection.
- Repository URL: The URL from where a running instance of Oracle API Catalog can be accessed.
- User Name: The user name for the Oracle API Catalog.
- Password: The password for the Oracle API Catalog.

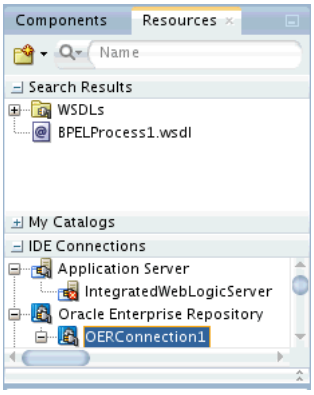
5. Click **Test Connection**. A success message is displayed in the Status pane.
6. Click **OK**.
7. In the Resource Palette, under IDE Connections, expand Oracle Enterprise Repository to see the application server connection that you created, as shown in Figure 3–3.

Figure 3–3 Resource Palette



8. Enter a search criteria to search for assets in the Search field. A list of all the assets is displayed, as shown in Figure 3–4.

Figure 3–4 Search Results



3.3 Edit an Existing Repository Connection

To edit an existing connection between JDeveloper and Oracle API Catalog:

1. In the Resource Palette, under IDE Connections, expand Oracle Enterprise Repository to see the existing repository connections.

2. Right-click on the connection you want to edit, and then click **Properties**.
3. Edit any of the following fields:
   - Repository URL: The URL from where a running instance of Oracle API Catalog can be accessed.
   - User Name: The user name for the Oracle API Catalog.
   - Password: The password for the Oracle API Catalog.
4. Click **Test Connection**. A success message is displayed in the Status pane.
5. Click **OK** to save the changes and close the dialog.

4

Using Oracle JDeveloper to Interact with Oracle API Catalog

This chapter describes the consumption processes and the interaction of Oracle JDeveloper with Oracle API Catalog.

This chapter includes the following sections:

- Using Oracle JDeveloper

4.1 Using Oracle JDeveloper

The Oracle API Catalog provides a flexible meta model for cataloging all assets within the SOA ecosystem. It is primarily used during the plan, design, and build phase of the lifecycle as a single source of truth for application development.

You can use Oracle SOA Suite with Oracle API Catalog 12*c*.

This section describes the following topics:

- Section 4.1.1, "Associating a JDeveloper Application with Oracle API Catalog"
- Section 4.1.2, "Search Oracle API Catalog"
- Section 4.1.3, "Consuming an API from Oracle Enterprise Repository"

4.1.1 Associating a JDeveloper Application with Oracle API Catalog

You must associate the JDeveloper application with a default Oracle API Catalog connection to consume assets from Oracle API Catalog using JDeveloper. To associate a JDeveloper application with a default Oracle API Catalog connection:

1. In Oracle JDeveloper, expand the **Tools** menu, and then select **Oracle API Catalog**. The Configure Oracle Enterprise Repository Integration dialog is displayed.

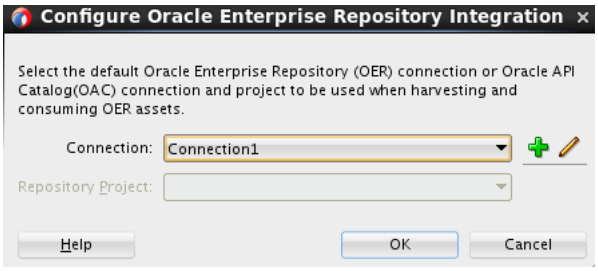
   Note: You select **Oracle Enterprise Repository** from JDeveloper's Tools Menu even when using Oracle API Catalog.

2. Select the following options, as shown in Figure 4–1.

   - Repository Connection: Select the Oracle API Catalog connection that you want to use for usage tracking.

     Note: The Repository Project option is not available for Oracle API Catalog.

Figure 4–1 Configure Oracle Enterprise Repository Integration dialog



This figure illustrates the Configure Oracle Enterprise Repository Integration dialog. The dialog displays the Connection and Repository Project lists. The Repository Project list is not available for OAC connections. The New And Edit icons appear next to the Connection list; these icons allow you to create a new repository connection or edit the connection selected in the Connections list. Help, OK, and Cancel buttons are also displayed.

****

3. Click **OK**.

You can now consume assets from the connection that you have selected.

4.1.2 Search Oracle API Catalog

You can access the APIs available in the Oracle Enterprise Repository through Oracle JDeveloper. Through Oracle JDeveloper, you can search for APIs matching various criteria or view APIs that may be of interest to a development project.

To search for assets in Oracle Enterprise Repository, perform the following steps:

1. In Oracle JDeveloper, click **Resource Palette**. The Resource Palette tab with the IDE Connections is displayed.
2. In the Search text field, enter the search criteria, for example, the name of the API that you want to view the details for, and click **Start Search**. The Search Results pane is displayed with the API.

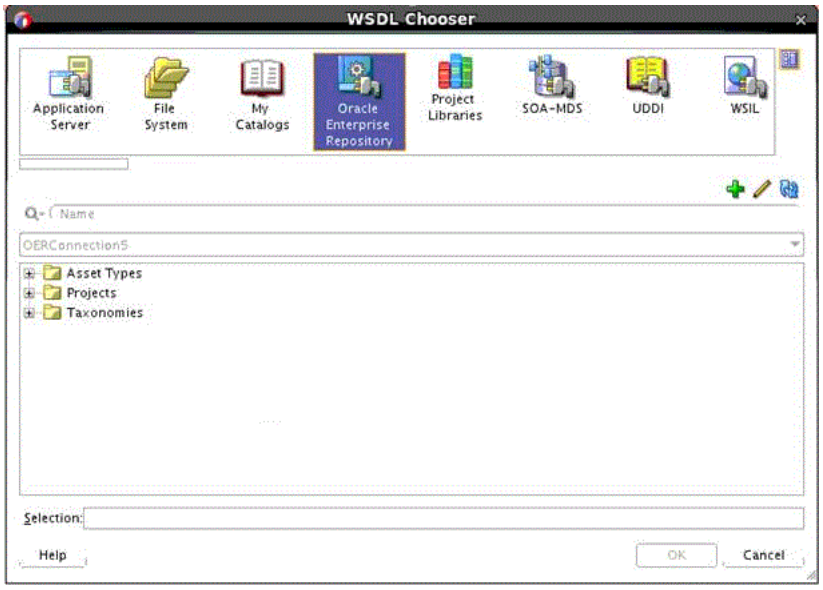
4.1.3 Consuming an API from Oracle Enterprise Repository

You can download an API into your project. Typically, an API's payload is a functionality that you need to use a service, such as a WSDL or WADL file.

To consume a WSDL file or a service from Oracle Enterprise Repository, perform the following steps:

1. In Oracle JDeveloper, double-click the **composite.xml** file. The composite.xml page is displayed.
2. Drag and drop the Web Service component from the Component Palette to the External References swim lane. The Create Web Service dialog is displayed.
3. Click the **Finding Existing WSDLs** icon at the end of the WSDL URL field. The WSDL Chooser dialog is displayed.
4. Select the Oracle Enterprise Repsoitory tab, as shown in Figure 4–2.

Figure 4–2 WSDL Chooser Dialog



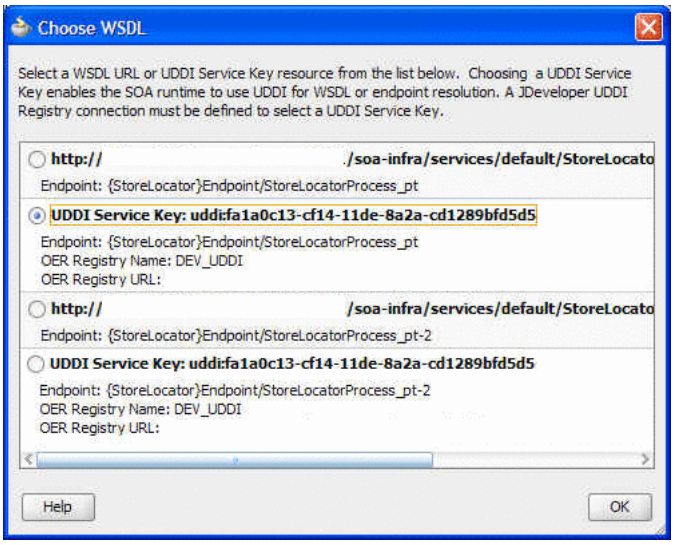
This figure illustrates the WSDL Chooser dialog. The Oracle Enterprise Repsoitory tab is selected.

****

5. Use the navigation tree to find the service that you want to invoke/consume.

   If the service has only one WSDL or UDDI key associated with it, then the same WSDL or UDDI key is used to create the reference. If service has more than one WSDL and/or UDDI key associated with it, then the Choose WSDL dialog is displayed, as shown in Figure 4–3.

Figure 4–3 Choose WSDL Dialog



This figure illustrates the Choose WSDL dialog.

****

You must select an URLs/UDDI keys to consume. For resolving UDDI keys in JDeveloper, you have to create a UDDI connection prior to creating the reference, without which you cannot select UDDI keys.

6. Click **OK**.

To consume a WADL file or a service from Oracle Enterprise Repository, perform the following steps:

1. In Oracle JDeveloper, double-click the **composite.xml** file. The composite.xml page is displayed.
2. Drag and drop the REST Service component from the Component Palette to the External References swim lane. The Create REST Binding dialog is displayed.
3. Enter the required information into the **Name**, **Type**, and **Base URI** fields.
4. Enter the necessary information into the **Resources** panel.
5. From the Operation Bindings panel, click the **Add** icon and select **Add operations based on WADL Service**. The WADL Chooser dialog is displayed.
6. Select the **Oracle Enterprise Repository** tab.
7. Use the navigation tree to find the service that you want to invoke/consume.

Part IV

Developing Custom Integrations Using the REX API

This part describes developing custom integrations for Oracle API Catalog using the REX API Framework.

This part contains the following chapter:

5

Using the Repository Extensibility Framework

This chapter describes the Repository Extensibility Framework (REX) architecture and discusses how to enable the OpenAPI and consume the WSDL.

This chapter includes the following sections:

5.1 Introduction to REX

REX is a web services API for programmatic integration into Oracle API Catalog. It is based on accepted industry standards, and designed with a focus on interoperability and platform independence. REX uses Remote Procedure Call (RPC) web services described by the Web Services Description Language (WSDL v1.1). This enables clients to interact with Oracle API Catalog using any platform and any implementation language that supports web services. For example, while Oracle API Catalog is a J2EE application, REX enables programmatic interaction with a .NET client.

> **Note:** Instances of "flashline" and "registry" appear in this documentation, particularly in the java package structure and in the REX class names.

If your Oracle API Catalog is or will be configured to be secured by Siteminder, you must configure the policy server to ignore (or unprotect) the following URL to allow OpenAPI integration to function properly:

http://appserver.example.com/oer/services/

> **Note:** The examples provided in the documentation are for illustrative purposes and will not necessarily compile due to package structure differences between versions of the Oracle API Catalog WSDL. You must change the package structure to appropriately target your version of Oracle API Catalog.

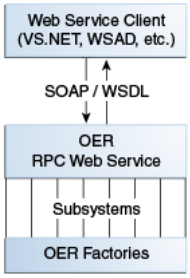
For more information about REX, see the REX Javadoc at *Oracle Fusion Middleware Extensibility Framework (REX) for Oracle API Catalog*.

Note: Not all of the REX features apply to Oracle API Catalog. The REX Javadoc contains features supported in Oracle Enterprise Repository, but not Oracle API Catalog. Oracle recommends that you use only the REX features collected in this document. An Oracle API Catalog license includes rights to use only a subset of the entire REX package set.

5.2 REX Architecture

Figure 5–1 describes the REX architecture.

Figure 5–1 REX Architecture



The high-level architecture of Oracle API Catalog and REX is designed with several high-level goals in mind:

- **Flexibility**

  Any client platform that conforms to accepted industry standards, such as SOAP, WSDL, and HTTP, can interact with Oracle API Catalog through the REX interface. Proper functioning of the API with the most common client platforms has been validated.

- **Extensibility**

  Oracle API Catalog's layered architecture simplifies the process of adding subsystems to provide access to new features as they are added to Oracle API Catalog. For more information, see Section 5.2.4, "Versioning Considerations for the Oracle API Catalog REX".

- **Simplicity**

  End users find it easy to take advantage of the extensive feature set available in REX.

5.2.1 Subsystems Overview

Oracle API Catalog's REX provides access to a variety of subsystems. These subsystems loosely group system functionality into logical categories roughly equivalent to the type of entity on which they operate. Much of this document is organized into sections related to these subsystems.

REX methods are named using a scheme based on the various subsystems. For more information about the description of the algorithm used in this process, see Section 5.2.2, "CRUD-Q Naming Convention". The subsystems defined in REX include:

- asset
- department

- import/export
- user

5.2.2 CRUD-Q Naming Convention

The scheme used in naming the Open API methods is based on the CRUD-Q mnemonic. CRUD-Q represents five operations:

- C - Create
- R - Read
- U - Update
- D - Delete
- Q - Query

Each method starts with the name of the subsystem to which it belongs, followed by a description of the operation to be performed within that subsystem, as in the following example:

<subsystem><Operation>

For example, the method to perform a create operation in the asset subsystem would be:

```
assetCreate(...)
```

This naming convention would also produce:

```
assetRead(...)
assetUpdate(...)
assetDelete(...)
assetQuery(...)
```

Subsystems are likely to have operations beyond the CRUD-Q set, and may not include all of CRUD-Q. For example, since it is impossible to delete a user, there is no userDelete method. There is, however, a userDeactivate method. Table 5–1 provides a detailed list of the detailed operations that the subsystem can have apart from the CRUD-Q operations.

Table 5–1 Subsystems and CRUD-Q Convention Relationship

	Create	Read	Update	Delete	Query	Other Features
Asset	Yes	Yes	Yes	Yes	Yes	
Department	Yes	Yes	Yes	No	Yes	
User	Yes	Yes	Yes	No	Yes	Activate, Deactivate, Lockout, Unapprove

5.2.2.1 Atomicity of Method Calls

Unless otherwise noted, every call to REX is atomic. That is, each call either succeeds completely, or fails completely.

For example, one version of the `categorizationUpdate` method takes as an argument an array of categorization updates. In this case, if one categorization update fails, all categorization updates fail.

5.2.2.2 No Inter-call Transaction Support

REX does not currently support inter-call transactions. For example, in the event of an error it is impossible to roll back operations associated with a series of REX calls.

5.2.3 Fundamental WSDL Data Types

REX uses the following fundamental WSDL data types, in addition to the complex types defined in the WSDL.

Arrays of any of these types can be returned:

- xsd:int
- xsd:long
- xsd:string
- xsd:boolean
- xsd:dateTime

You can dynamically generate API Stubs by consuming the REX WSDL by pointing its IDEs or web services toolkits at the following URL:

http://appserver/oer/services/FlashlineRegistry?WSDL

If you are creating custom integrations with OER from Oracle BPM 11g, use the following URL instead:

http://appserver/oer/services/RexAPI?wsdl

Java stubs for the Oracle API Catalog REX WSDL can be created using the AXIS WSDL2java utility:

```
java -cp .;axis.jar; xerces.jar; commons-discovery.jar;
commons-logging.jar; jaxen-full.jar; jaxrpc.jar; saaj.jar;wsdl4j.jar;
xalan.jar org.apache.axis.wsdl.WSDL2Java
```

The JAR files required to complete this conversion process are:

- `axis.jar`
- `xerces.jar`
- `commons-discovery.jar`
- `commons-logging.jar`
- `jaxen-full.jar`
- `jaxrpc.jar`
- `saaj.jar`
- `wsdl4j.jar`
- `xalan.jar`

> **Note:** Replace "appserver" in the URL with the name of the server on which Oracle API Catalog is installed.

5.2.4 Versioning Considerations for the Oracle API Catalog REX

The evolution of the Oracle API Catalog REX parallels the evolution of Oracle API Catalog. As a result of this process, incompatibilities may emerge between older and

newer versions of REX. While full version compatibility is our goal, backward compatibility is subject to unpredictable and therefore potentially unavoidable limitations. Oracle API Catalog REX includes the following backward compatible enhancements:

- Addition of new methods to the Oracle API Catalog web service.
- Definition of new complex types in the WSDL.

With regard to these backward compatible changes, the regeneration of client proxies is necessary only when the need arises to take advantage of new features and functionality.

The namespace of the service changes only when incompatible changes are unavoidable. Examples of such a change would include the modification of an existing complex type, or a change in the signature of a method in the service. In this event, client proxy regeneration is necessary, as are the minimal code changes. Client proxies generated from prior versions of REX are unable to connect to the new service.

The namespace of complex types never changes.

5.3 Basic Concepts

This section describes the basic concepts of REX such as getting started with enabling the OpenAPI and consuming the WSDL.

5.3.1 Enabling the OpenAPI within the Oracle API Catalog

The procedure is performed on the Oracle API Catalog Admin screen.

1. Click **System Settings**.
2. Enter the property cmee.extframework.enabled in the Enable New System Setting text box.
3. Click **Enable**. The Open API section is displayed.
4. Ensure the **cmee.extframework.enabled** property is set to `True`.
5. Click **Save**. REX is now enabled within your instance of Oracle API Catalog.

5.3.2 Consuming WSDL

The first step in using REX is to generate the client-side stubs necessary to communicate with the Oracle API Catalog server. This is generally accomplished using the automated tools provided by the specific Web services toolkit in use. This section describes how to generate client stubs using a variety of integrated development environments and toolkits.

Authentication and Authorization

The first step in using REX is authenticating with the server. Authentication is performed using the authTokenCreate method. This method takes a user ID and password as arguments to be used in authenticating with Oracle API Catalog. If the ID and password are successfully authenticated, an authentication token is returned. This token must be used in every subsequent call to REX.

If a valid AuthToken is not included for every REX method, an OpenAPIException is thrown. The applies to all methods except authTokenCreate and authTokenDelete.

The following example shows how to retrieve an AuthToken and use it in subsequent REX calls.

Example 5–1 How to Retrieve an AuthToken and use in REX Calls

```
package com.example.flashlineclient;
//The imports below are assumed for any of the included examples
import javax.xml.rpc.ServiceException;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import com.flashline.registry.openapi.service.v300.FlashlineRegistry;
import
 com.flashline.registry.openapi.service.v300.FlashlineRegistryServiceLocator;
import com.flashline.registry.openapi.base.OpenAPIException;
import com.flashline.registry.openapi.entity.AuthToken;
import com.flashline.registry.openapi.entity.Asset;
public class FlexTest {
public FlexTest () {
}
public static void main(String[] pArgs)throws OpenAPIException, RemoteException,
ServiceException {
try {
FlashlineRegistry lRegistry = null;
AuthToken lAuthToken = null;
URL lURL = null;
lURL = new URL("http://www.example.com/appname/services/FlashlineRegistry");
//"www.example.com" should be your server address
//"appname" is the application name of the location that the Registry is running
 on
//These two things must be changed to the proper values in every example
lRegistry = new FlashlineRegistryServiceLocator().getFlashlineRegistry(lURL);
lAuthToken = lRegistry.authTokenCreate("username", "password");
System.out.println(lAuthToken.getToken());
//displaying the authtoken as a string to the screen
Asset lAsset = lRegistry.assetRead(lAuthToken, 559);
//reading asset number 559
System.out.println(lAsset.getName());
//displaying the name of asset 559 to the screen
} catch(OpenAPIException lEx) {
System.out.println("ServerCode = "+ lEx.getServerErrorCode());
System.out.println("Message = "+ lEx.getMessage());
System.out.println("StackTrace:");
lEx.printStackTrace();
} catch (RemoteException lEx) {
lEx.printStackTrace();
} catch (ServiceException lEx) {
lEx.printStackTrace();
} catch (MalformedURLException lEx) {
lEx.printStackTrace();
}
System.out.println("execution completed");
System.exit(0);
}
}
```

Authorization

REX enforces the same authorization rules as the Oracle API Catalog application. The user ID and password used to authenticate determines the privileges available to the user through REX. For example, if the authenticated user does not have EDIT privileges assigned for projects, and attempts to create a project using the projectCreate REX method, an OpenAPIException is thrown.

Exception Handling

Open API communicates server errors to the client through a SOAP Fault. The manner in which SOAP Faults are handled varies according the language and SOAP toolkit in use.

This section suggests ways to detect and deal with exceptions generated by the Open API within client code, using the most common platform/toolkit combinations.

Java and AXIS

Exceptions thrown by the Open API are transferred as SOAP Faults, and then deserialized by the AXIS client toolkit as Java Exceptions. That is, AXIS makes an attempt to map the SOAP Fault to a corresponding client-side OpenAPIException class. Server-side errors are represented to the client as `com.flashline.registry.openapi.OpenAPIException` instances. Consequently, client code can catch exceptions with the code listed below which is from the code above:

```
try {
lAsset = lRegistry.assetCreate(..);
} catch(OpenAPIException lEx) {
System.out.println("ServerCode = "+ lEx.getServerErrorCode());
System.out.println("Message = "+ lEx.getMessage());
System.out.println("StackTrace:");
lEx.printStackTrace();
} catch (RemoteException lEx) {
lEx.printStackTrace();
} catch (ServiceException lEx) {
lEx.printStackTrace();
} catch (MalformedURLException lEx) {
lEx.printStackTrace();
}
```

Validation

When attempting to save an entity in Oracle API Catalog, the system attempts to validate the input. Any missing or invalid data causes the server to throw an OpenAPIException containing a list of fields and their respective errors.

For more information, see "Exception Handling".

Query Considerations in REX

The criteria object model is currently moving to a more flexible representation of terms and grouping. As they occur, these changes affects the availability of certain API features when executing a query using a criteria object. The subsystems only directly evaluate their corresponding criteria objects, and do not make use of the extended capabilities of the underlying SearchTermGroup, unless otherwise noted in this documentation.

Sending Binary Data (Attachments)

Various REX methods require sending or receiving potentially large sets of binary data between the client and server. For example, the import/export subsystem provides methods for sending a payload from which to import, and methods for retrieving a payload representing a set of exported assets.

Typically, binary data is transferred through Web services RPC invocations through Dynamic Internet Message Exchange (DIME); SOAP with Attachments (SwA); or Base-64 Encoding. Each has its advantages and disadvantages, but few client toolkits directly support all three.

The Oracle API Catalog OpenAPI supports all three mechanisms for transferring binary data. Details are provided in the following sections. Any method that provides for the binary transfer of data has three versions, each one supporting a different transfer mechanism. For example, to retrieve the results of an export, a user can select any one of the following methods:

- **importGetResultsB64**

  Retrieve results of export in base-64 encoded format. This is the lowest common denominator, and can be used on any platform, provided that the client can encode/decode base-64 data.

- **importGetResultsDIME**

  Retrieve export results as an attached file, using the DIME protocol. This is the preferred option for most .NET clients.

- **importGetResultsSwA**

  Retrieve export results as an attached file, using the SOAP with Attachments (SwA) protocol (MIME-based)

Using DIME attachments with .NET and the Microsoft Web Services Enhancement (WSE) Kit

Microsoft provides an extension to the standard .Net Web service toolkit. The Microsoft Web Services Enhancement (WSE) kit provides advanced functionality, such as sending and receiving attachments through Web services using the Dynamic Internet Messaging Exchange (DIME) protocol.

The following code snippet gives an example of sending data through a DIME attachment:

Example 5–2 Example of Sending Data Through a DIME Attachment

```
// relax the requirement for the server to understand ALL headers. This MUST be
// done, or the call with the attachment fails. After the call, if you wish,
// you can set this back to "true"
registry.RequestSoapContext.Path.EncodedMustUnderstand= "false";
// clear the attachments queue
registry.RequestSoapContext.Attachments.Clear();
registry.RequestSoapContext.Attachments.Add(
new Microsoft.Web.Services.Dime.DimeAttachment("0", "application/zip",
Microsoft.Web.Services.Dime.TypeFormatEnum.MediaType, "c:\\tmp\\import.zip"));
// start an import running on the server
registry.importExecute(lAuthToken, "flashline", null, "FEA Flashpack Import",
null);
// do some polling (calls to importStatus) to monitor the import progress,
// if you wish
```

The following code snippet gives an example of receiving data through a DIME attachment:

Example 5–3 Example of Receiving Data Through a DIME Attachment

```
// relax the requirement for the server to understand ALL headers. This MUST be
// done, or the call with the attachment fails. After the call, if you wish,
// you can set this back to "true"
registry.RequestSoapContext.Path.EncodedMustUnderstand= "false";
// clear the attachments queue
registry.RequestSoapContext.Attachments.Clear();
// start an export
```

```
flashline.ImpExpJob lJob =
registry.exportExecute(lAuthToken, "flashline", null, "Complete Export",
 "flashline",
"<entitytypes>
<entitytype type=\"acceptableValueList\">
<entities>
<entity id=\"100\"/>
</entities>
</entitytype>
</entitytypes>");
// do some polling (calls to exportStatus) to watch the progress of the
// export, if you wish...
// this call blocks until either the method returns (or an exception is
 thrown),
// or the call times out.
registry.exportGetResultsDIME(lAuthToken, lJob);
// check to see if the call resulted in attachments being returned...
if(registry.ResponseSoapContext.Attachments.Count > 0)
{
Stream lStream = registry.ResponseSoapContext.Attachments[0].Stream;
// write the data out somewhere...
}
```

Using SOAP with Attachments and Java AXIS clients

The Axis client provides functions to handle SOAP attachments in Java. For more information, see
http://www-106.ibm.com/developerworks/webservices/library/ws-soapatt/

The following code snippet gives an example of receiving data:

```
byte[] lResults = null;
ImpExpJob lExportJob =
mFlashlineRegistrySrc.exportExecute(mAuthTokenSrc,"flashline",null,
"Export Assets","default", createAssetQuery().toString());
lExportJob =
mFlashlineRegistrySrc.exportStatus(mAuthTokenSrc, lExportJob);
lResults =
mFlashlineRegistrySrc.exportGetResultsB64(mAuthTokenSrc, lExportJob);
// write the results out to disk in a temp file
File lFile = null;
String lTempDirectory =
System.getProperty("java.io.tmpdir");
lFile = new File(lTempDirectory + File.separator + "impexp.zip");
FileOutputStream lOS = new FileOutputStream(lFile);
BufferedOutputStream lBOS = new BufferedOutputStream(lOS);
lBOS.write(lResults);
lBOS.flush();
lBOS.close();
lOS.close();
```

The following code snippet gives an example of sending data through a DIME attachment:

```
// open file and attach as data source
InputStream lIS = new FileInputStream(lFile);
((Stub)mFlashlineRegistryDest)._setProperty
(Call.ATTACHMENT_ENCAPSULATION_FORMAT, Call.ATTACHMENT_ENCAPSULATION_FORMAT_DIME);
ByteArrayDataSource lDataSource = new ByteArrayDataSource(lIS,
 "application/x-zip-compressed");
DataHandler lDH = new DataHandler(lDataSource);
```

```
// add the attachment
((Stub)mFlashlineRegistryDest).addAttachment(lDH);
ImpExpJob lJob =
mFlashlineRegistryDest.importExecute(mAuthTokenDest, "flashline", null, "Import
 Assets Test", null);
```

5.4 REX API Descriptions and Use Cases

This section describes and provides use cases for the REX API.

> **Note:** Not all of the REX features apply to Oracle API Catalog. The REX Javadoc contains features supported in Oracle Enterprise Repository, but not Oracle API Catalog. Oracle recommends that you use only the REX features collected in this document.

5.4.1 Asset API

This section describes the Asset API and provides use cases for the Asset API that describe how to create or modify an asset, build an asset search, and work with asset status and asset tabs.

5.4.1.1 Overview

Assets are the core entities in the Oracle API Catalog. This document covers the Asset subsystem actions Create, Read, Update, Delete, and Query. It also covers the modification of:

- Asset active status:
  - Activate
  - Deactivate
  - Retire
- Asset registration status:
  - Submit
  - Accept
  - Register
  - Unsubmit
  - Unaccept
  - Unregister
- Asset assignment status:

  - Assign
  - Unassign

Several issues must be taken into consideration when working with assets in Oracle API Catalog using REX. Trade-offs in memory consumption and performance should be carefully weighed.

- **Memory Consumption**

  Assets and their metadata consume a significant amount of memory on both the REX server and on the client software using REX. When searching Oracle API Catalog using REX, it is possible that a large number of assets may be returned. To avoid Denial of Service interruptions to the system, administrators can limit the maximum number of results that can be returned in a call to the REX method assetQuery. The system setting cmee.extframework.assets.max controls the maximum number of search results that can be returned as a result of a query. If the number of results matching a query exceeds the maximum, an exception is generated by REX.

  In cases where it is expected that a potentially large number of assets matches a query, the assetQuerySummary method is recommended. This alternative method of querying Oracle API Catalog matches exactly the same assets as a call to assetQuery, but returns lightweight asset summary objects, rather than the full asset objects. These summary objects consume a nominal amount of memory, and the possibility of exhausting resources as a result of a query is consequently negligible.

  After a summary query has been performed, the full asset objects can be retrieved for assets of interest using either assetRead, or assetReadArrayFromSummary. If multiple assets are desired, use of the assetReadArrayFromSummary method is recommended. See the API documentation for details on using this method.

- **Performance**

  REX is based on standard web services technology, which provides many significant advantages in flexibility and portability. However, as with any web services-based technology, performance can be challenging, particularly in high data volume situations (for example, large numbers of assets being manipulated). REX provides options that allow developers to avoid potential performance problems.

  - **Iterative Reads**

    The primary overhead in web services technology is incurred in the serialization and de-serialization of data using XML, combined with network transfer. Much of this overhead can be avoided in situations where a number of assets are to be read. For example, if 50 assets are to be retrieved from Oracle API Catalog using REX, the developer could perform 50 assetRead calls. A better approach, however, would be to use the assetReadArray method, passing the IDs of the desired assets as a single argument. This would retrieve all 50 assets in one call, dramatically improving performance.

  - **Listing Operations**

    Often data is retrieved from REX for the purpose of displaying a listing to an end user, who then is expected to select an asset for closer inspection. In cases like these, the full extent of asset metadata is not required to generate a summary list. As discussed in the section on memory above, consider using the summary methods provided in REX.

- **Access Control**

- Which assets a user of REX can see, and to some extent the information in those assets, is controlled by access settings. The same access restrictions that exist for a user accessing the system through the web GUI also apply to the REX asset subsystem.
- Query restrictions - users can only retrieve assets in a call to assetQuery or assetRead for which they have view permission.
- Update restrictions - users can only update assets for which they have edit permission.
- File restrictions - users can only view the files for which they have download permissions as set in the File type Custom Access Settings applied to each individual file. This means that a user might be able to view an asset, but might not be able to view any of the asset's files. Each file can have it's own permissions, different from the asset's permissions. If specific File type permissions are not applied to a file, these permissions are inherited from the asset's permissions to which the files belong.

5.4.1.1.1 Definitions

This section provides the following definitions:

- **ID and UUID**
  - ID is an internal unique identifier (numeric) used to identify an asset uniquely within a single Oracle API Catalog instance.
  - UUID is a universally unique identifier (128-bit numeric represented as a hexidecimal string) used to identify an asset uniquely across any instance of Oracle API Catalog. Each asset's UUID is exposed primarily for purposes of reading and searching. Oracle strongly advises not modifying this field using REX. However, if an administrator does choose to modify an asset's UUID, then the format must be consistent (00000000-0000-0000-0000-000000000000) and the UUID must be unique within the target Oracle API Catalog instance; otherwise, the change operation fails.
- **Name and Version**

  String fields that combine to uniquely identify an asset.
- **Custom Data**

  Customizable metadata for an asset is stored in an XML structure within this string. The sample code describes the custom data methods effectively.

5.4.1.1.2 Sample Code

Example 5–4 Sample Code for Custom Data Methods

```
package com.flashline.sample.assetapi;
import java.io.StringReader;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import java.text.Format;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;
import javax.xml.rpc.ServiceException;
import org.jdom.Document;
import org.jdom.Element;
import org.jdom.input.SAXBuilder;
import org.jdom.output.XMLOutputter;
import org.jdom.xpath.XPath;
import com.flashline.registry.openapi.base.OpenAPIException;
```

```
import com.flashline.registry.openapi.entity.AuthToken;
import com.flashline.registry.openapi.service.v300.FlashlineRegistry;
import
 com.flashline.registry.openapi.service.v300.FlashlineRegistryServiceLocator;
public class UpdateCustomData {
  public static void main(String pArgs[]) throws OpenAPIException,
 RemoteException, ServiceException {
    try {
      ////////////////////////////////////////////////////////
      // Connect to Oracle API Catalog
      ////////////////////////////////////////////////////////
      URL lURL = null;
      lURL = new URL(pArgs[0]);
      FlashlineRegistry repository = new FlashlineRegistryServiceLocator()
      .getFlashlineRegistry(lURL);
      ////////////////////////////////////////////////////////
      // Login to OER
      ////////////////////////////////////////////////////////
      AuthToken authToken = repository.authTokenCreate(
          pArgs[1],pArgs[2]);
      ////////////////////////////////////////
      // assetUpdateCustomDataString
      ////////////////////////////////////////
      // Find and Modify a custom data field value with a value supplied on the
 command line
      Long currentLicenses = new
 Long(repository.assetReadCustomDataString(authToken, 589, "/
_-total-licenses-owned"));
      currentLicenses = new Long(currentLicenses.intValue() + 1);
      //    save the modifications
      //  NOTICE: A leading slash is required to specify the appropriate path to
 the element being updated.
      repository.assetUpdateCustomDataString(authToken, 589, "/
_-total-licenses-owned", currentLicenses.toString());
      ////////////////////////////////////////
      // assetUpdateCustomDataStringArray
      ////////////////////////////////////////
      // Add a custom data field value with a value supplied on the command line
      Format formatter = new SimpleDateFormat("yyyyMMdd");
      String dateFormat = formatter.format(new Date());
      // NOTICE: for the following method, there is no leading slash for the
 elements being updated.
      String[] versionHistory = {"version-history/version-history/version-number",
 "version-history/version-history/production-date-yyyymmdd-",
 "version-history/version-history/comments"};
      String[] versionHistoryValues = {currentLicenses.toString(), dateFormat,
 "Updated version History: " + dateFormat};
      //    save the modifications
      repository.assetUpdateCustomDataStringArray(authToken, 589, versionHistory,
 versionHistoryValues);
      ////////////////////////////////////////
      // assetUpdateCustomDataNode
      ////////////////////////////////////////
      //The following updates a specific custom data element called
 "document-name" that is a child of "document",
      //which is a child of "documentation"
      XPath lXPath = null;
      List lElms = null;
      //First read the Node "documentation" of the specific asset
      String lXMLDocumentation = repository.assetReadCustomDataNode(authToken,
```

```
589, "documentation");
     //Using DOM, convert the XML to a Document
     Document lDoc = null;
     SAXBuilder lBuilder = new SAXBuilder();
     StringReader lReader = null;
     lReader = new StringReader(lXMLDocumentation);
     lBuilder.setValidation(false);
     lBuilder.setIgnoringElementContentWhitespace(true);
     lDoc = lBuilder.build(lReader);
     lXPath = XPath.newInstance("documentation/document");
     lElms = lXPath.selectNodes(lDoc);
     //Cycle through the "document" elements until we find the one we want.  Then
update it.
     for (int i=0;i<lElms.size();i++) {
       Element lElm = (Element)lElms.get(i);
       List lChildElms = lElm.getChildElements();
       for (int x=0;x<lChildElms.size();x++) {
         Element lChildElm = (Element)lChildElms.get(x);
         if (lChildElm.getName().equals("document-name") &&
lChildElm.getValue().equals("API")) {
           lChildElm.setText("API KHAN");
         } else {
           lChildElm.setText(lChildElm.getValue());
         }
       }
     }
     //Convert the Document back to an XML string and update the asset's custom
data.
     repository.assetUpdateCustomDataNode(authToken, 589, "documentation", new
XMLOutputter().outputString(lDoc));
     ////////////////////////////////////////
     // assetUpdateCustomDataNodeArray
     ////////////////////////////////////////
     try {
       //The following updates multiple custom data elements.  One is called
"document-name" that is a child of "document",
       //which is a child of "documentation"
       //The other is the element called "also-known-as"
       lXPath = null;
       lElms = null;
       //First read the Node "documentation" of the specific asset
       lXMLDocumentation = repository.assetReadCustomDataNode(authToken, 589,
"documentation");
       //Using DOM, convert the XML to a Document
       lDoc = null;
       lBuilder = new SAXBuilder();
       lReader = null;
       lReader = new StringReader(lXMLDocumentation);
       lBuilder.setValidation(false);
       lBuilder.setIgnoringElementContentWhitespace(true);
       lDoc = lBuilder.build(lReader);
       lXPath = XPath.newInstance("documentation/document");
       lElms = lXPath.selectNodes(lDoc);
       //Cycle through the "document" elements until we find the one we want.
Then update it.
       for (int i=0;i<lElms.size();i++) {
         Element lElm = (Element)lElms.get(i);
         List lChildElms = lElm.getChildElements();
         for (int x=0;x<lChildElms.size();x++) {
           Element lChildElm = (Element)lChildElms.get(x);
```

```
            if (lChildElm.getName().equals("document-name") &&
 lChildElm.getValue().equals("API")) {
              lChildElm.setText("API KHAN");
            } else {
              lChildElm.setText(lChildElm.getValue());
            }
          }
        }
        String lDoc1 = new XMLOutputter().outputString(lDoc);
        //Get the next element
        lXMLDocumentation = repository.assetReadCustomDataNode(authToken, 589,
 "also-known-as");
        lDoc = null;
        lBuilder = new SAXBuilder();
        lReader = null;
        lReader = new StringReader(lXMLDocumentation);
        lBuilder.setValidation(false);
        lBuilder.setIgnoringElementContentWhitespace(true);
        lDoc = lBuilder.build(lReader);
        lXPath = XPath.newInstance("also-known-as");
        lElms = lXPath.selectNodes(lDoc);
        //Get the also-known-as element
        for (int i=0;i<lElms.size();i++) {
          Element lElm = (Element)lElms.get(i);
          lElm.setText("Modified Alias");
        }
        String lDoc2 = new XMLOutputter().outputString(lDoc);
        //Convert the Document back to an XML string and update the asset's custom
 data.
        repository.assetUpdateCustomDataNodeFromStringArray(authToken, 589, new
 String[] {"documentation", "also-known-as"}, new String[] {lDoc1, lDoc2});
      } catch (Exception e) {
        e.printStackTrace();
      }
    } catch (OpenAPIException lEx) {
      System.out.println("ServerCode = " + lEx.getServerErrorCode());
      System.out.println("Message = " + lEx.getMessage());
      System.out.println("StackTrace:");
      lEx.printStackTrace();
    } catch (RemoteException lEx) {
      lEx.printStackTrace();
    } catch (ServiceException lEx) {
      lEx.printStackTrace();
    } catch (MalformedURLException lEx) {
      lEx.printStackTrace();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

5.4.1.1.3 Related Subsystems This sections describes the following related subsystems that are used with the Asset API:

- **AssetType**

  All assets need an active and valid asset type. This asset type defines the metadata that can be stored in the custom data for the asset.

- **Vendor**

  If desired, an asset may be linked to a vendor. This linking is done by the vendor ID.

- **AcceptableValueLists**

  When creating or editing assets, acceptable values contained in acceptable value lists are used to as options for the metadata elements that were defined for the asset type. To use the acceptable values for an acceptable value list, modify the custom data for the asset (Asset.GetCustomData()) to have it reference the ID of the acceptable value.

- **RelationshipType**

  Relationship types define the kinds of relationships that can exist between assets.

- **Categorization Types**

  Categorization types are top-level groups of categorizations added to asset types. Categorizations describe an asset.

- **Projects**

  Assets can be produced by projects. The producing projects for an asset are stored in an array of ID's.

- **Users**

  Users can be assigned to assets. They are the person who is responsible for working up the metadata.

5.4.1.2 Use Cases

This section describes the use cases using the Asset API. It includes the following topics:

5.4.1.2.1 Use Case: Creating a New Asset Description

Create a new asset and enter it into Oracle API Catalog.

Sample Code

Example 5–5 Use Case: Creating a New Asset

```
package com.flashline.sample.assetapi;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import java.util.Calendar;
import javax.xml.rpc.ServiceException;
import com.flashline.registry.openapi.base.OpenAPIException;
import com.flashline.registry.openapi.entity.Asset;
import com.flashline.registry.openapi.entity.AuthToken;
import com.flashline.registry.openapi.service.v300.FlashlineRegistry;
import
 com.flashline.registry.openapi.service.v300.FlashlineRegistryServiceLocator;
public class CreateNewAsset {
  public static void main(String pArgs[]) throws OpenAPIException,
 RemoteException,
      ServiceException {
    try {
      ////////////////////////////////////////////////////////////
      // Connect to Oracle API Catalog
      ////////////////////////////////////////////////////////////
      URL lURL = null;
      lURL = new URL(pArgs[0]);
      FlashlineRegistry repository = new FlashlineRegistryServiceLocator()
          .getFlashlineRegistry(lURL);
      ////////////////////////////////////
      // Login to OER
      ////////////////////////////////////
      AuthToken authToken = repository.authTokenCreate(
          pArgs[1],pArgs[2]);
      ////////////////////////////////////
      // Create new asset
      ////////////////////////////////////
      Asset myAsset = repository.assetCreate(authToken,
          "My Asset Name108", "My Version
 "+Calendar.getInstance().getTimeInMillis(), 144);
      ////////////////////////////////////////////
      //The following demonstrates how to modify a custom data Date element on an
 asset.
      //This date must be in a specific format and the name of the element must by
 known.
      //In this example, the name of the element is "testdate".  This element must
 have been created in the
      //asset type as a Date element
      //Update the testdate field to January 1, 2007
      //Note: the format of the date should match the system setting for Short
 Date.
      repository.assetUpdateCustomDataString(authToken, myAsset.getID(),
 "testdate", "2007-1-1");
    } catch (OpenAPIException lEx) {
      System.out.println("ServerCode = " + lEx.getServerErrorCode());
      System.out.println("Message    = " + lEx.getMessage());
      System.out.println("StackTrace:");
      lEx.printStackTrace();
```

```
    } catch (RemoteException lEx) {
      lEx.printStackTrace();
    } catch (ServiceException lEx) {
      lEx.printStackTrace();
    } catch (MalformedURLException lEx) {
      lEx.printStackTrace();
    }
  }
}
```

5.4.1.2.2 Use Case: Creating a New Asset from XML Description

It is also possible to create a new asset from an XML representation of the asset. Schemas are used to validate the asset XML before creation. The schema for an asset type is available through the Open API as can be seen in Example 5–6.

It is not necessary to do validation yourself, the asset XML is validated internally before the create happens. If you do want to do your own validation, then you must find a validating XML parser such as Xerces 2.0.

Sample Code

Example 5–6 Use Case: Creating a New Asset from XML

```
package com.flashline.sample.assetapi;
import java.io.IOException;
import java.io.StringReader;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import java.util.Calendar;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.rpc.ServiceException;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.helpers.DefaultHandler;
import com.flashline.registry.openapi.base.OpenAPIException;
import com.flashline.registry.openapi.entity.AuthToken;
import com.flashline.registry.openapi.service.v300.FlashlineRegistry;
import
 com.flashline.registry.openapi.service.v300.FlashlineRegistryServiceLocator;
public class CreateNewAssetFromXML {
  public static void main(String pArgs[]) throws ServiceException,
      ParserConfigurationException, SAXException, IOException {
    String SCHEMA_LANGUAGE =
 "http://java.sun.com/xml/jaxp/properties/schemaLanguage";
    String XML_SCHEMA = "http://www.w3.org/2001/XMLSchema";
    String SCHEMA_SOURCE = "http://java.sun.com/xml/jaxp/properties/schemaSource";
    SAXParserFactory lSaxParserFactory = null;
    SAXParser lSaxParser = null;
    try {
      ////////////////////////////////////////////////////////
      // Connect to Oracle API Catalog
      ////////////////////////////////////////////////////////
      URL lURL = null;
      lURL = new URL(pArgs[0]);
      FlashlineRegistry repository = new FlashlineRegistryServiceLocator()
```

```
            .getFlashlineRegistry(lURL);
        ////////////////////////////////////
        // Login to OER
        ////////////////////////////////////
        AuthToken authToken = repository.authTokenCreate(pArgs[1],pArgs[2]);
        // Anonymous class to handle validation errors
        DefaultHandler lDefaultHandler = new DefaultHandler() {
          public void error(SAXParseException exception) throws SAXException {
            throw exception;
          }
          public void fatalError(SAXParseException exception) throws SAXException {
            throw exception;
          }
        };
        ////////////////////////////////////////////////////////////////////
        // Define the asset you want to create in XML
        ////////////////////////////////////////////////////////////////////
        // This is the XML of the asset we're creating. Typically it would
        // come from a GUI or other asset creation mechanism. It is hard
        // coded for this example.
        ////////////////////////////////////////////////////////////////////
        String assetXML = "<asset id=\"0\">"
            + "           <asset-type id=\"145\" icon=\"component.gif\"
lastSavedDate=\"17 Jul 2007 12:00:00 AM\">Component</asset-type>"
            + "           <mandatory-data>"
            + "               <name>NewComponent</name>"
            + "
<version>"+Calendar.getInstance().getTimeInMillis()+"</version>"
            + "               <description><![CDATA[My Description]]></description>"
            + "               <keywords/>"
            + "               <notification-email/>"
            + "               <applied-policies/>"
            + "               <vendor id=\"0\"/>"
            + "               <file-informations/>"
            + "               <hash-informations/>"
            + "               <producing-projects/>"
            + "               <submission-files/>"
            + "               <applied-compliance-templates/>"
            + "               <contacts/>"
            + "               <relationships/>"
            + "               <categorization-types/>"
            + "           </mandatory-data>"
            + "           <admin-data>"
            + "           </admin-data>"
            + "       </asset>";
        ////////////////////////////////////////////////////////////////////
        // This returns the Schema for the asset type of the asset we're
        // creating
        ////////////////////////////////////////////////////////////////////
        String schema = repository.assetTypeSchemaRead(authToken, 144);
        ////////////////////////////////////////////////////////////////////
        // This block of code shows validating the asset XML against
        // the schema
        ////////////////////////////////////////////////////////////////////
        lSaxParserFactory = SAXParserFactory.newInstance();
        lSaxParserFactory.setNamespaceAware(true);
        lSaxParserFactory.setValidating(true);
        lSaxParser = lSaxParserFactory.newSAXParser();
        lSaxParser.setProperty(SCHEMA_LANGUAGE, XML_SCHEMA);
        lSaxParser.setProperty(SCHEMA_SOURCE, new InputSource(new StringReader(
```

```
          schema)));
      lSaxParser.parse(new InputSource(new StringReader(assetXML)),
          lDefaultHandler);
      ////////////////////////////////////////////////////////////////////
      // If no exception was thrown the asset XML validates and
      // the creation should not fail due to XML formatting errors.
      ////////////////////////////////////////////////////////////////////
      repository.assetCreateFromXML(authToken, assetXML);
    } catch (OpenAPIException lEx) {
      System.out.println("ServerCode = " + lEx.getServerErrorCode());
      System.out.println("Message    = " + lEx.getMessage());
      System.out.println("StackTrace:");
      lEx.printStackTrace();
    } catch (RemoteException lEx) {
      lEx.printStackTrace();
    } catch (ServiceException lEx) {
      lEx.printStackTrace();
    } catch (MalformedURLException lEx) {
      lEx.printStackTrace();
    }
  }
}
```

5.4.1.2.3 Use Case: Modifying an Asset Description

Modify the metadata for an existing asset.

Sample Code

Example 5–7 Use Case: Modifying an Asset

```
package com.flashline.sample.assetapi;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import com.flashline.registry.openapi.base.OpenAPIException;
import com.flashline.registry.openapi.entity.Asset;
import com.flashline.registry.openapi.entity.AuthToken;
import com.flashline.registry.openapi.entity.Categorization;
import com.flashline.registry.openapi.entity.CategorizationType;
import com.flashline.registry.openapi.query.CategorizationTypeCriteria;
import com.flashline.registry.openapi.service.v300.FlashlineRegistry;
import
 com.flashline.registry.openapi.service.v300.FlashlineRegistryServiceLocator;
public class ModifyExistingAsset {
  public static void main(String pArgs[]) throws OpenAPIException,
 RemoteException,
      ServiceException {
    try {
      ////////////////////////////////////////////////////////////
      // Connect to Oracle API Catalog
      ////////////////////////////////////////////////////////////
      URL lURL = null;
      lURL = new URL(pArgs[0]);
      FlashlineRegistry repository = new FlashlineRegistryServiceLocator()
          .getFlashlineRegistry(lURL);
      ////////////////////////////////////////////////////////////
      // Login to OER
      ////////////////////////////////////////////////////////////
```

```
      AuthToken authToken = repository.authTokenCreate(
          pArgs[1],pArgs[2]);
      ////////////////////////////////////////////////////////////
      // Read the asset you want to modify
      ////////////////////////////////////////////////////////////
      Asset myAsset = repository.assetRead(authToken, 559);
      // 559 is the example asset number
      ////////////////////////////////////////////////////////////
      // Modify the name, version, description, and notification
      // email
      ////////////////////////////////////////////////////////////
      myAsset.setName("New Name");
      myAsset.setVersion("New Version");
      myAsset.setDescription("New Description");
      myAsset.setNotificationEmail("user@example.com");
      ////////////////////////////////////////////////////////////
      // Modify Categorizations on the asset
      ////////////////////////////////////////////////////////////
      // Setup arrays used for assigning categorizations
      CategorizationType[] lAllCatTypes = null;
      Categorization[] lAllCats = null;
      CategorizationType[] lCatTypes = new CategorizationType[1];
      Categorization[] lCats = new Categorization[1];
      ////////////////////////////////////////////////////////////
      // Search for all categorizations that are asset assignable
      ////////////////////////////////////////////////////////////
      CategorizationTypeCriteria categorizationTypeCriteria = new
CategorizationTypeCriteria();
      categorizationTypeCriteria.setNameCriteria("");
      lAllCatTypes = repository.categorizationTypeQuery(authToken,
          categorizationTypeCriteria);
      ////////////////////////////////////////////////////////////
      // Find all the categorizations to be assigned to the asset
      ////////////////////////////////////////////////////////////
      for (int i = 0; i < lAllCatTypes.length; i++) {
        CategorizationType lCatType = repository.categorizationTypeRead(
            authToken, lAllCatTypes[i].getID());
        lAllCats = repository.categorizationReadByType(authToken,
            lCatType, true, true);
        if (lAllCats.length > 0) {
          lCatTypes[0] = lCatType;
          // when we find the first one, use it
          break;
        }
      }
      lCats[0] = lAllCats[0];
      ////////////////////////////////////////////////////////////
      // Modify the asset to use the categorizations
      ////////////////////////////////////////////////////////////
      myAsset.setCategorizations(lCats);
      myAsset.setCategorizationTypes(lCatTypes);
      ////////////////////////////////////////////////////////////
      // Modify the custom access settings for the asset
      ////////////////////////////////////////////////////////////
      String[] lCasTypes = repository.customAccessSettingTypesGet(authToken);
      String[] lCustomAccessSettings = null;
      if (lCasTypes!=null && lCasTypes.length>0) {
        lCustomAccessSettings = repository.customAccessSettingNamesGet(authToken,
lCasTypes[0]);
      }
```

```
      if (lCustomAccessSettings!=null && lCustomAccessSettings.length>0) {
        String[] myCustomAccessSettings = { lCustomAccessSettings[0] };
        myAsset.setCustomAccessSettings(myCustomAccessSettings);
      }
      ////////////////////////////////////////////////////////
      // Add producing projects to the asset
      ////////////////////////////////////////////////////////
      long[] producingProjectsIDs = new long[1];
      producingProjectsIDs[0] = 50000;
      myAsset.setProducingProjectsIDs(producingProjectsIDs);
      ////////////////////////////////////////////////////////
      // save the modifications
      ////////////////////////////////////////////////////////
      repository.assetUpdate(authToken, myAsset);
    } catch (OpenAPIException lEx) {
      System.out.println("ServerCode = " + lEx.getServerErrorCode());
      System.out.println("Message    = " + lEx.getMessage());
      System.out.println("StackTrace:");
      lEx.printStackTrace();
    } catch (RemoteException lEx) {
      lEx.printStackTrace();
    } catch (ServiceException lEx) {
      lEx.printStackTrace();
    } catch (MalformedURLException lEx) {
      lEx.printStackTrace();
    }
  }
}
```

5.4.1.2.4 Use Case: Assign Users to an Asset Description

Multiple users can be assigned to an asset.

Sample Code

Example 5–8 Use Case: Assigning Users to an Asset

```
package com.flashline.sample.assetapi;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import java.util.Calendar;
import javax.xml.rpc.ServiceException;
import com.flashline.registry.openapi.base.OpenAPIException;
import com.flashline.registry.openapi.entity.Asset;
import com.flashline.registry.openapi.entity.AssignedUser;
import com.flashline.registry.openapi.entity.AuthToken;
import com.flashline.registry.openapi.entity.RegistryUser;
import com.flashline.registry.openapi.query.UserCriteria;
import com.flashline.registry.openapi.service.v300.FlashlineRegistry;
import
 com.flashline.registry.openapi.service.v300.FlashlineRegistryServiceLocator;
public class AssignUsers {
  public static void main(String pArgs[]) throws OpenAPIException,
 RemoteException,
      ServiceException {
    try {
      ////////////////////////////////////////////////////////
      // Connect to Oracle API Catalog
      ////////////////////////////////////////////////////////
```

```
  URL lURL = null;
  lURL = new URL(pArgs[0]);
  FlashlineRegistry repository = new FlashlineRegistryServiceLocator()
      .getFlashlineRegistry(lURL);
  ////////////////////////////////////////////////////////
  // Login to OER
  ////////////////////////////////////////////////////////
  AuthToken authToken = repository.authTokenCreate(
      pArgs[1],pArgs[2]);
  ////////////////////////////////////////////////////////
  // Retrieve desired asset
  ////////////////////////////////////////////////////////
  Asset myAsset = repository.assetRead(authToken, 559);
  // 559 is the example asset number
  ////////////////////////////////////////////////////////
  // Create array of AssignedUser objects
  ////////////////////////////////////////////////////////
  AssignedUser[] lUsers = new AssignedUser[3];
  ////////////////////////////////////////////////////////
  // NOTE:
  //
  // The AssignedUser object has two methods:
  // setUserID(long)
  // setAssignedDate(Calendar).
  // (Specifies the date the user was assigned to the
  // asset. If no date is specified, the current date
  // is used.)
  ////////////////////////////////////////////////////////
  ////////////////////////////////////////////////////////
  // Add AssignedUser objects to the array
  ////////////////////////////////////////////////////////
  AssignedUser lUser = new AssignedUser();
  lUser.setUserID(99); // 99 is the admin user id
  lUsers[0] = lUser;
  lUser = new AssignedUser();
  RegistryUser lRegistryUser1 = createRegistryUser(repository, authToken);
  lUser.setUserID(lRegistryUser1.getID());
  lUsers[1] = lUser;
  lUser = new AssignedUser();
  RegistryUser lRegistryUser2 = createRegistryUser(repository, authToken);
  lUser.setUserID(lRegistryUser2.getID());
  lUsers[2] = lUser;
  ////////////////////////////////////////////////////////
  // Add array to the asset that is being updated
  ////////////////////////////////////////////////////////
  myAsset.setAssignedUsers(lUsers);
  ////////////////////////////////////////////////////////
  // save the modifications
  ////////////////////////////////////////////////////////
  repository.assetUpdate(authToken, myAsset);
} catch (OpenAPIException lEx) {
  System.out.println("ServerCode = " + lEx.getServerErrorCode());
  System.out.println("Message    = " + lEx.getMessage());
  System.out.println("StackTrace:");
  lEx.printStackTrace();
} catch (RemoteException lEx) {
  lEx.printStackTrace();
} catch (ServiceException lEx) {
  lEx.printStackTrace();
} catch (MalformedURLException lEx) {
```

```
      lEx.printStackTrace();
    }
  }
  protected static RegistryUser createRegistryUser(FlashlineRegistry repository,
 AuthToken authToken) throws OpenAPIException, RemoteException {
    RegistryUser lRet = null;
    String lUserName = "user"+Calendar.getInstance().getTimeInMillis();
    lRet = repository.userCreate(authToken, lUserName, "", lUserName,
 lUserName+"@example.com", lUserName, false, false, false);
    return lRet;
  }
}
```

5.4.1.2.5 Use Case: Building an Asset Search Description

Finding all assets that meet certain criteria.

Sample Code

Example 5–9 Use Case: Building an Asset Search

```
package com.flashline.sample.assetapi;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import com.flashline.registry.openapi.base.OpenAPIException;
import com.flashline.registry.openapi.entity.AssetSummary;
import com.flashline.registry.openapi.entity.AuthToken;
import com.flashline.registry.openapi.query.AssetCriteria;
import com.flashline.registry.openapi.query.DateRangeSearchTerm;
import com.flashline.registry.openapi.query.SearchTerm;
import com.flashline.registry.openapi.query.TabStatusSearchTerm;
import com.flashline.registry.openapi.service.v300.FlashlineRegistry;
import
 com.flashline.registry.openapi.service.v300.FlashlineRegistryServiceLocator;
public class FindAssets {
  public static void main(String pArgs[]) throws OpenAPIException,
 RemoteException,
      ServiceException {
    try {
      ////////////////////////////////////////////////////////
      // Connect to Oracle API Catalog
      ////////////////////////////////////////////////////////
      URL lURL = null;
      lURL = new URL(pArgs[0]);
      FlashlineRegistry repository = new FlashlineRegistryServiceLocator()
          .getFlashlineRegistry(lURL);
      ////////////////////////////////////////////////////////
      // Login to OER
      ////////////////////////////////////////////////////////
      AuthToken authToken = repository.authTokenCreate(
          pArgs[1],pArgs[2]);
      ////////////////////////////////////////////////////////
      // Search for all assets
      ////////////////////////////////////////////////////////
      AssetCriteria criteria = new AssetCriteria();
      AssetSummary[] assets = repository.assetQuerySummary(authToken, criteria);
      ////////////////////////////////////////////////////////
      // try a general search which includes Name, version,
```

```
     // description, and keywords
     ////////////////////////////////////////////////////////////
     criteria = new AssetCriteria();
     criteria.setGeneralCriteria("My Asset");
     assets = repository.assetQuerySummary(authToken, criteria);
     ////////////////////////////////////////////////////////////
     // Search for assets that contain a specific search string
     // in one particular field.
     ////////////////////////////////////////////////////////////
     criteria = new AssetCriteria();
     criteria.setNameCriteria("My Name");
     criteria.setVersionCriteria("My version");
     criteria.setDescriptionCriteria("My Description");
     assets = repository.assetQuerySummary(authToken, criteria);
     ////////////////////////////////////////////////////////////
     // Implementing a Search through the AssetCriteria Object
     // -----------------------------------------------
     // If no operator is specified when implementing a search
     // using the setSearchTerms method in the AssetCriteria
     // object, the system defaults to the operator EQUALS.
     // The operator LIKE must be specified if required for the
     // search.
     ////////////////////////////////////////////////////////////
     criteria = new AssetCriteria();
     SearchTerm lSearchTerm = new SearchTerm();
     lSearchTerm.setKey("name");
     lSearchTerm.setOperator("LIKE");
     lSearchTerm.setValue("Test");
     SearchTerm[] lTerms = new SearchTerm[1];
     lTerms[0] = lSearchTerm;
     criteria.setSearchTerms(lTerms);
     assets = repository.assetQuerySummary(authToken, criteria);
     //Search for assets where a specific DATETIME field is a given age
     ////////////////////////////////////////////////////////////////
     criteria = new AssetCriteria();
     //Not specifying an operator defaults to 'equals'.
     DateRangeSearchTerm lTerm = new DateRangeSearchTerm();
     ////////////////////////
     //date-range is the query key.  registereddate is the date field we are
searching on
     //Allowable fields to search on:
     //submitteddate
     //registereddate
     //accepteddate
     //createddate
     //updateddate
     //To do a search for all assets that were registered more than 5 days ago
     lTerm.setKey("date-range");
     //Set the value to a day 5 days older than the current date.  Assume today's
date is 1/10/2007
     //The format defaults to OER's system setting for Short Date Format.
     //The format can be set to any valid date format using setDateFormat() and
passing the date format.
     lTerm.setDateFormat("yyyy-MM-dd");
     lTerm.setBeginDate("2007-1-5");
     lTerm.setBeginOperator("lt");
     lTerm.setDateField("registereddate");
     lTerms = new SearchTerm[1];
     lTerms[0] = lTerm;
     criteria.setSearchTerms(lTerms);
```

```
     assets = repository.assetQuerySummary(authToken, criteria);
     //Search for assets where a given date field is within a date range
     ////////////////////////////////////////////////////////////////
     criteria = new AssetCriteria();
     lTerm = new DateRangeSearchTerm();
     ////////////////////////
     //date-range is the query key.  registereddate is the date field we are
searching on
     //Allowable fields to search on:
     //submitteddate
     //registereddate
     //accepteddate
     //createddate
     //updateddate
     //The following SearchTerm translates to "Assets where the registereddate is
greater than or equal to Jan. 1, 2007
     lTerm.setKey("date-range");
     //The format defaults to OER's system setting for Short Date Format.
     //The format can be set to any valid date format using setDateFormat() and
passing the date format.
     lTerm.setDateField("registereddate");
     lTerm.setDateFormat("yyyy-MM-yy");
     lTerm.setBeginDate("2007-01-01");
     lTerm.setBeginOperator("gte");
     lTerm.setEndDate("2007-01-10");
     lTerm.setEndOperator("lte");
     //The following SearchTerm translates to "Assets where the registereddate is
less than or equal to Jan. 10, 2007
     criteria.setSearchTerms(new SearchTerm[] {lTerm});
     //This query returns all assets that were registered between January 1 and
January 10, including those days.
     assets = repository.assetQuerySummary(authToken, criteria);
     //Search for assets where a given tab has been approved or unapproved
     ////////////////////////////////////////////////////////////////
     criteria = new AssetCriteria();
     TabStatusSearchTerm lTabTerm = new TabStatusSearchTerm();
     //tabstatus is the type of search we want to do.  overview is the name of
the tab we want to search on
     lTabTerm.setKey("tabstatus");
     lTabTerm.setTabNames(new String[] {"overview"});
     lTabTerm.setApproved(true);
     criteria.setSearchTerms(new SearchTerm[] {lTabTerm});
     //This query returns all assets with the Overview tab being approved
     assets = repository.assetQuerySummary(authToken, criteria);
     //You may also search by a date range.
     lTabTerm.setKey("tabstatus");
     lTabTerm.setTabNames(new String[] {"overview"});
     lTabTerm.setApproved(false);
     lTabTerm.setBeginDate("2007-1-01");
     lTabTerm.setBeginOperator("lte");
     criteria.setSearchTerms(new SearchTerm[] {lTabTerm});
     //The following returns all assets that have the Overview tab unapproved
since or before January 1, 2007
     assets = repository.assetQuerySummary(authToken, criteria);
     //Search for assets where a custom field date has a specific value.
     //This test returns all assets that have a testdate of January 1, 2007.
     //The testdate field is a custom data date element.
     ////////////////////////////////////////////////////////////////
     criteria = new AssetCriteria();
     DateRangeSearchTerm lDateRangeTerm = new DateRangeSearchTerm();
```

```
      //Test Equals
      lDateRangeTerm.setKey("/asset/custom-data/testdate");
      lDateRangeTerm.setBeginDate("2007-01-1");
      lDateRangeTerm.setBeginOperator("eq");
      criteria.setSearchTerms(new SearchTerm[] {lDateRangeTerm});
      assets = repository.assetQuerySummary(authToken, criteria);
    } catch (OpenAPIException lEx) {
      System.out.println("ServerCode = " + lEx.getServerErrorCode());
      System.out.println("Message    = " + lEx.getMessage());
      System.out.println("StackTrace:");
      lEx.printStackTrace();
    } catch (RemoteException lEx) {
      lEx.printStackTrace();
    } catch (ServiceException lEx) {
      lEx.printStackTrace();
    } catch (MalformedURLException lEx) {
      lEx.printStackTrace();
    }
  }
}
```

5.4.1.2.6 Use Case: Upgrading Asset Status Description

Stepping an asset through status levels, from Unsubmitted to Submitted to Accepted to Registered.

Sample Code

Example 5–10 Use Case: Upgrading Asset Status

```
package com.flashline.sample.assetapi;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import com.flashline.registry.openapi.base.OpenAPIException;
import com.flashline.registry.openapi.entity.AuthToken;
import com.flashline.registry.openapi.entity.KeyValuePair;
import com.flashline.registry.openapi.query.AssetCriteria;
import com.flashline.registry.openapi.service.v300.FlashlineRegistry;
import
 com.flashline.registry.openapi.service.v300.FlashlineRegistryServiceLocator;
public class PromoteAsset {
  public static void main(String pArgs[]) throws OpenAPIException,
 RemoteException,
      ServiceException {
    try {
      ////////////////////////////////////////////////////////
      // Connect to Oracle API Catalog
      ////////////////////////////////////////////////////////
      URL lURL = null;
      lURL = new URL(pArgs[0]);
      FlashlineRegistry repository = new FlashlineRegistryServiceLocator()
          .getFlashlineRegistry(lURL);
      ////////////////////////////////////////////////////////
      // Login to OER
      ////////////////////////////////////////////////////////
      AuthToken authToken = repository.authTokenCreate(
          pArgs[1],pArgs[2]);
      long lAssetID = 559;
```

```
      // ------------------------------------------------------
      // asset with id 559 would have to be unsubmitted for this to work
      AssetCriteria lAssetCriteria = new AssetCriteria();
      lAssetCriteria.setIDCriteria(lAssetID);
      KeyValuePair lKeyValuePair = repository.assetEvaluate(authToken,
 lAssetCriteria, "Registration Status");
      if (!lKeyValuePair.getValue().equalsIgnoreCase("unsubmitted")) {
        unregisterAsset(repository, authToken, lAssetID);
      }
      ////////////////////////////////////////////////////////
      // promote the asset from unsubmitted to submitted
      ////////////////////////////////////////////////////////
      repository.assetSubmit(authToken, lAssetID);
      // asset 559 would have to be unsubmitted for this to work
      ////////////////////////////////////////////////////////
      // promote the asset from submitted to accepted
      ////////////////////////////////////////////////////////
      repository.assetAccept(authToken, lAssetID);
      // asset 561 would have to be submitted for this to work
      ////////////////////////////////////////////////////////
      // promote the asset from accepted to registered
      ////////////////////////////////////////////////////////
      repository.assetRegister(authToken, lAssetID);
      // asset 563 would have to be accepted for this to work
    } catch (OpenAPIException lEx) {
      System.out.println("ServerCode = " + lEx.getServerErrorCode());
      System.out.println("Message    = " + lEx.getMessage());
      System.out.println("StackTrace:");
      lEx.printStackTrace();
    } catch (RemoteException lEx) {
      lEx.printStackTrace();
    } catch (ServiceException lEx) {
      lEx.printStackTrace();
    } catch (MalformedURLException lEx) {
      lEx.printStackTrace();
    }
  }
  protected static void unregisterAsset(FlashlineRegistry repository, AuthToken
 authToken, long pAssetID) {
    try {
      repository.assetUnRegister(authToken, pAssetID);
    } catch (Exception e) {
    }
    try {
      repository.assetUnAccept(authToken, pAssetID);
    } catch (Exception e) {
    }
    try {
      repository.assetUnSubmit(authToken, pAssetID);
    } catch (Exception e) {
    }
  }
}
```

5.4.1.2.7 Use Case: Downgrading Asset Status Description

The reverse of the previous use case, stepping an asset through status levels, from Registered to Accepted to Submitted to Unsubmitted.

Sample Code

Example 5–11 Use Case: Downgrading Asset Status

```
package com.flashline.sample.assetapi;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import com.flashline.registry.openapi.base.OpenAPIException;
import com.flashline.registry.openapi.entity.AuthToken;
import com.flashline.registry.openapi.entity.KeyValuePair;
import com.flashline.registry.openapi.query.AssetCriteria;
import com.flashline.registry.openapi.service.v300.FlashlineRegistry;
import
 com.flashline.registry.openapi.service.v300.FlashlineRegistryServiceLocator;
public class DemoteAsset {
  public static void main(String pArgs[]) throws OpenAPIException,
 RemoteException,
      ServiceException {
    try {
      ////////////////////////////////////////////////////////////
      // Connect to Oracle API Catalog
      ////////////////////////////////////////////////////////////
      URL lURL = null;
      lURL = new URL(pArgs[0]);
      FlashlineRegistry repository = new FlashlineRegistryServiceLocator()
          .getFlashlineRegistry(lURL);
      ////////////////////////////////////////////////////////////
      // Login to OER
      ////////////////////////////////////////////////////////////
      AuthToken authToken = repository.authTokenCreate(
          pArgs[1],pArgs[2]);
      long lAssetID = 559;
      // ------------------------------------------------------
      // asset with id 559 would have to be registered for this to work
      AssetCriteria lAssetCriteria = new AssetCriteria();
      lAssetCriteria.setIDCriteria(lAssetID);
      KeyValuePair lKeyValuePair = repository.assetEvaluate(authToken,
 lAssetCriteria, "Registration Status");
      if (!lKeyValuePair.getValue().equalsIgnoreCase("registered")) {
        registerAsset(repository, authToken, lAssetID);
      }
      ////////////////////////////////////////////////////////////
      // demote the asset from registered to accepted
      ////////////////////////////////////////////////////////////
      repository.assetUnRegister(authToken, lAssetID);
      ////////////////////////////////////////////////////////////
      // demote the asset from accepted to submitted
      ////////////////////////////////////////////////////////////
      repository.assetUnAccept(authToken, lAssetID);
      ////////////////////////////////////////////////////////////
      // demote the asset from submitted to unsubmitted
      ////////////////////////////////////////////////////////////
      repository.assetUnSubmit(authToken, lAssetID);
    } catch (OpenAPIException lEx) {
      System.out.println("ServerCode = " + lEx.getServerErrorCode());
      System.out.println("Message    = " + lEx.getMessage());
      System.out.println("StackTrace:");
      lEx.printStackTrace();
```

```
    } catch (RemoteException lEx) {
      lEx.printStackTrace();
    } catch (ServiceException lEx) {
      lEx.printStackTrace();
    } catch (MalformedURLException lEx) {
      lEx.printStackTrace();
    }
  }
  protected static void registerAsset(FlashlineRegistry repository, AuthToken
 authToken, long pAssetID) {
    try {
      repository.assetSubmit(authToken, pAssetID);
    } catch (Exception e) {
    }
    try {
      repository.assetAccept(authToken, pAssetID);
    } catch (Exception e) {
    }
    try {
      repository.assetRegister(authToken, pAssetID);
    } catch (Exception e) {
    }
  }
}
```

5.4.1.2.8 Use Case: Apply and Remove Compliance Templates from a Project Description

Compliance Templates can be added and removed from multiple projects.

> **Note:** An OpenAPIException occurs if an asset is applied to a project and that asset is NOT a Compliance Template.

Sample Code

Example 5–12 Use Case: Applying and Removing Compliance Templates from a Project

```
package com.flashline.sample.assetapi;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import java.util.Calendar;
import javax.xml.rpc.ServiceException;
import com.flashline.registry.openapi.base.OpenAPIException;
import com.flashline.registry.openapi.entity.Asset;
import com.flashline.registry.openapi.entity.AssetType;
import com.flashline.registry.openapi.entity.AuthToken;
import com.flashline.registry.openapi.entity.Project;
import com.flashline.registry.openapi.query.AssetCriteria;
import com.flashline.registry.openapi.query.AssetTypeCriteria;
import com.flashline.registry.openapi.service.v300.FlashlineRegistry;
import
 com.flashline.registry.openapi.service.v300.FlashlineRegistryServiceLocator;
public class AddRemoveTemplate {
  public static void main(String pArgs[]) throws OpenAPIException,
 RemoteException,
      ServiceException {
    try {
      URL lURL = null;
```

```
      lURL = new URL(pArgs[0]);
      ////////////////////////////////////////////////////////////
      // Connect to Oracle API Catalog
      ////////////////////////////////////////////////////////////
      FlashlineRegistry repository = new FlashlineRegistryServiceLocator()
          .getFlashlineRegistry(lURL);
      ////////////////////////////////////////////////////////////
      // Login to OER
      ////////////////////////////////////////////////////////////
      AuthToken authToken = repository.authTokenCreate(pArgs[1],
          pArgs[2]);
      ////////////////////////////////////////////////////////////
      // Read or Create a Compliance Template Type and Asset
      ////////////////////////////////////////////////////////////
      AssetType ctType = null;
      AssetTypeCriteria lAssetTypeCriteria = new AssetTypeCriteria();
      lAssetTypeCriteria.setArcheTypeCriteria("Compliance Template Type");
      AssetType[] lAssetTypes =
        repository.assetTypeQuery(authToken, lAssetTypeCriteria);
      if (lAssetTypes!=null && lAssetTypes.length>0) {
        ctType = lAssetTypes[0];
      } else {
        ctType = repository.assetTypeCreateComplianceTemplate(authToken,
            "My Compliance Template
Type"+Calendar.getInstance().getTimeInMillis());
      }
      Asset lComplianceTemplateAsset = null;
      AssetCriteria lAssetCriteria = new AssetCriteria();
      lAssetCriteria.setAssetTypeCriteria(ctType.getID());
      Asset[] lAssets = repository.assetQuery(authToken, lAssetCriteria);
      if (lAssets!=null && lAssets.length>0) {
        lComplianceTemplateAsset = lAssets[0];
      } else {
        lComplianceTemplateAsset = repository.assetCreate(authToken, "My
Compliance Template",
            ""+Calendar.getInstance().getTimeInMillis(), ctType.getID());
      }
      ////////////////////////////////////////////////////////////
      // Create a String array of Project IDs that the Compliance
      // Template is applied to.
      ////////////////////////////////////////////////////////////
      String[] lProjectIDs = { "50000" };
      // ////////////////////////////////////////////////////////
      // Apply template to the projects.
      // ////////////////////////////////////////////////////////
      repository.assetApplyToProjects(authToken, lProjectIDs,
          lComplianceTemplateAsset);
      ////////////////////////////////////////////////////////////
      // Retrieve an array of Projects that this template is
      // applied to.
      ////////////////////////////////////////////////////////////
      Project[] lProjects = repository.assetReadAppliedToProjects(
          authToken, lComplianceTemplateAsset);
      String lMsg = "Compliance Template '" + lComplianceTemplateAsset.getName();
      lMsg += "' applied to Project(s): ";
      for (int i=0; lProjects!=null && i<lProjects.length; i++) {
        lMsg += ""+lProjects[i].getName()+(i+1==lProjects.length ? "." : ", ");
      }
      System.out.println(lMsg);
      ////////////////////////////////////////////////////////////
```

```
      // Create a String array of Project IDs that the Compliance
      // Template is removed from.
      ////////////////////////////////////////////////////////
      String[] lRemoveProjectIDs = { "50000" };
      ////////////////////////////////////////////////////////
      // Remove template from the projects.
      ////////////////////////////////////////////////////////
      repository.assetRemoveAppliedToProjects(authToken,
          lRemoveProjectIDs, lComplianceTemplateAsset);
    } catch (OpenAPIException lEx) {
      System.out.println("ServerCode = " + lEx.getServerErrorCode());
      System.out.println("Message    = " + lEx.getMessage());
      System.out.println("StackTrace:");
      lEx.printStackTrace();
    } catch (RemoteException lEx) {
      lEx.printStackTrace();
    } catch (ServiceException lEx) {
      lEx.printStackTrace();
    } catch (MalformedURLException lEx) {
      lEx.printStackTrace();
    }
  }
}
```

5.4.1.2.9 Use Case: Creating the New Version of an Asset and Retiring the Old Version

Description

Update the repository to reflect the availability of a new version of an asset, and the retirement of the asset's previous version.

Sample Code

Example 5–13 Use Case: Creating a New Version of an Asset

```
package com.flashline.sample.assetapi;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import java.util.Calendar;
import javax.xml.rpc.ServiceException;
import com.flashline.registry.openapi.base.OpenAPIException;
import com.flashline.registry.openapi.entity.Asset;
import com.flashline.registry.openapi.entity.AuthToken;
import com.flashline.registry.openapi.entity.RelationshipType;
import com.flashline.registry.openapi.query.RelationshipTypeCriteria;
import com.flashline.registry.openapi.service.v300.FlashlineRegistry;
import
 com.flashline.registry.openapi.service.v300.FlashlineRegistryServiceLocator;
public class CreateNewVersionOfAsset {
  public static void main(String pArgs[]) throws OpenAPIException,
 RemoteException,
      ServiceException {
    try {
      ////////////////////////////////////////////////////////
      // Connect to Oracle API Catalog
      ////////////////////////////////////////////////////////
      URL lURL = null;
      lURL = new URL(pArgs[0]);
```

```
      FlashlineRegistry repository = new FlashlineRegistryServiceLocator()
          .getFlashlineRegistry(lURL);
      ////////////////////////////////////////////////////////////
      // Login to OER
      ////////////////////////////////////////////////////////////
      AuthToken authToken = repository.authTokenCreate(
          pArgs[1],pArgs[2]);
      ////////////////////////////////////////////////////////////
      // Read old asset.
      // Update metadata as necessary.
      // Save as new asset.
      ////////////////////////////////////////////////////////////
      Asset myAsset = repository.assetRead(authToken, 561);
      ////////////////////////////////////////////////////////////
      // Find the "next-version" relationship for the asset
      ////////////////////////////////////////////////////////////
      RelationshipType[] allRelationshipTypes =
getAllRelationshipTypes(repository, authToken);
      for (int i = 0; i < allRelationshipTypes.length; i++) {
        if (allRelationshipTypes[i].getName().equals("next-version")) {
          ////////////////////////////////////////////////////////
          // This is the relationship type, modify the assets that are related
          // using it
          ////////////////////////////////////////////////////////
          RelationshipType myRelationshipType = allRelationshipTypes[i];
          ////////////////////////////////////////////////////////
          // Add the old version to list of previous versions of the
          // newly created asset
          ////////////////////////////////////////////////////////
          long[] oldSecondaryIDs = myRelationshipType.getSecondaryIDs();
          long[] newSecondaryIDs = new long[oldSecondaryIDs.length + 1];
          for (int j = 0; j < oldSecondaryIDs.length; j++) {
            newSecondaryIDs[j] = oldSecondaryIDs[j];
          }
          newSecondaryIDs[newSecondaryIDs.length - 1] = 561;
          myRelationshipType.setSecondaryIDs(newSecondaryIDs);
        }
      }
      Asset myNewAsset = repository.assetCreate(authToken,
          myAsset.getName(), ""+Calendar.getInstance().getTimeInMillis(),
myAsset.getTypeID());
      myNewAsset.setRelationshipTypes(allRelationshipTypes);
      ////////////////////////////////////////////////////////////
      // Update the new asset
      ////////////////////////////////////////////////////////////
      myNewAsset = repository.assetUpdate(authToken, myNewAsset);
      ////////////////////////////////////////////////////////////
      // retire the old asset
      ////////////////////////////////////////////////////////////
      repository.assetRetire(authToken, 561);
    } catch (OpenAPIException lEx) {
      System.out.println("ServerCode = " + lEx.getServerErrorCode());
      System.out.println("Message    = " + lEx.getMessage());
      System.out.println("StackTrace:");
      lEx.printStackTrace();
    } catch (RemoteException lEx) {
      lEx.printStackTrace();
    } catch (ServiceException lEx) {
      lEx.printStackTrace();
    } catch (MalformedURLException lEx) {
```

```
      lEx.printStackTrace();
    }
  }
  /**
   * This method returns every relationship type in the repository
   * @param repository
   * @param authToken
   * @return
   * @throws RemoteException
   */
  public static RelationshipType[] getAllRelationshipTypes(FlashlineRegistry
 repository, AuthToken authToken) throws RemoteException {
    //Create an empty relationship type criteria object
    RelationshipTypeCriteria criteria = new RelationshipTypeCriteria();
    criteria.setNameCriteria("");
    RelationshipType[] allRelationshipTypes =
 repository.relationshipTypeQuery(authToken, criteria);
    return allRelationshipTypes;
  }
}
```

5.4.1.2.10 Use Case: Deleting Groups of Assets Description

Deleting groups of assets that no longer belong in the repository.

Sample Code

Example 5–14 Use Case: Deleting Unneeded Assets from the Repository

```
package com.flashline.sample.assetapi;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import com.flashline.registry.openapi.base.OpenAPIException;
import com.flashline.registry.openapi.entity.Asset;
import com.flashline.registry.openapi.entity.AuthToken;
import com.flashline.registry.openapi.query.AssetCriteria;
import com.flashline.registry.openapi.service.v300.FlashlineRegistry;
import
 com.flashline.registry.openapi.service.v300.FlashlineRegistryServiceLocator;
public class DeleteAssets {
  public static void main(String pArgs[]) throws OpenAPIException,
 RemoteException,
      ServiceException {
    try {
      ////////////////////////////////////////////////////////
      // Connect to Oracle API Catalog
      ////////////////////////////////////////////////////////
      URL lURL = null;
      lURL = new URL(pArgs[0]);
      FlashlineRegistry repository = new FlashlineRegistryServiceLocator()
          .getFlashlineRegistry(lURL);
      ////////////////////////////////////////////////////////
      // Login to OER
      ////////////////////////////////////////////////////////
      AuthToken authToken = repository.authTokenCreate(
          pArgs[1],pArgs[2]);
      ////////////////////////////////////////////////////////
      // find the assets to delete
```

```
      ////////////////////////////////////////////////////////
      AssetCriteria criteria = new AssetCriteria();
      criteria.setGeneralCriteria("delete me");
      Asset[] assets = repository.assetQuery(authToken, criteria);
      ////////////////////////////////////////////////////////
      // Iterate through assets, deleting them one at a time.
      ////////////////////////////////////////////////////////
      for (int i = 0; i < assets.length; i++) {
        repository.assetDelete(authToken, assets[i].getID());
      }
    } catch (OpenAPIException lEx) {
      System.out.println("ServerCode = " + lEx.getServerErrorCode());
      System.out.println("Message    = " + lEx.getMessage());
      System.out.println("StackTrace:");
      lEx.printStackTrace();
    } catch (RemoteException lEx) {
      lEx.printStackTrace();
    } catch (ServiceException lEx) {
      lEx.printStackTrace();
    } catch (MalformedURLException lEx) {
      lEx.printStackTrace();
    }
  }
}
```

Pitfalls:

Asset deletion is permanent. The OpenAPI provides no method for restoring deleted assets.

Methods to Avoid:

The following methods serve no purpose in the context of the OpenAPI, and should therefore be avoided:

- setAcceptedByID
- setAcceptedByName
- setAcceptedByDate
- setActiveStatus
- setAssigned
- setAssignedToID
- setAssignedDate
- setCategorizationTypes
- setCreatedByID
- setCreatedByName
- setCreatedByDate
- setDeleted
- setEntityType
- setExtractable
- setFullAsset
- setInactive

- setKey
- setLoadedDate
- setLongName
- setNotifyUpdatedRelationships
- setRegisteredByID
- setRegisteredByName
- setRegisteredDate
- setRegistrationStatus
- setRegistrationStatusBaseName
- setRegistrationStatusRegistered
- setRegistrationStatusRejected
- setRegistrationStatusSubmittedPendingReview
- setRegistrationStatusSubmittedUnderReview
- setRegistrationStatusUnsubmitted
- setRejectionReason
- setRetired
- setSubmittedByID
- setSubmittedByName
- setSubmittedDate
- setTypeIcon
- setTypeName
- setUpdatedDate
- setVendorName
- setVisible

Avoiding Common Mistakes

- Rules for Assets
  - The Asset must be assigned to an active and valid Asset Type.
  - An Asset's name/version strings must be a unique pair.
  - A new Asset's ID must be 0.
  - A new Asset's active status must be 'active'.

Missing Features

- Helper methods for modifying customData
- Additional validation

  When saving an asset, Oracle API Catalog currently validates that:
  - The Asset type is valid and active
  - # The Name/Version is unique
  - When creating an asset, that the active status is valid

- When updating an asset, that the asset already exists
- Contacts are not duplicated
- Categorizations are valid
- Future versions of the repository validates that:
  * CustomData is well formed XML
  * CustomData contains XML that is valid based on the asset type

5.4.1.2.11 Use Case: Finding Assets and Updating Custom-Data Description

Perform a search for all assets with a specific custom-data value, and update some custom-data for each of those assets. Note: The asset is automatically saved when using the assetUpdateCustomDataNode method.

Sample Code

Example 5–15 Use Case: Finding Assets and Updating Common-Data

```
package com.flashline.sample.assetapi;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import com.flashline.registry.openapi.base.OpenAPIException;
import com.flashline.registry.openapi.entity.AssetSummary;
import com.flashline.registry.openapi.entity.AuthToken;
import com.flashline.registry.openapi.query.AssetCriteria;
import com.flashline.registry.openapi.query.SearchTerm;
import com.flashline.registry.openapi.service.v300.FlashlineRegistry;
import
 com.flashline.registry.openapi.service.v300.FlashlineRegistryServiceLocator;
public class UpdateAssetTestResults {
  public static void main(String pArgs[]) throws OpenAPIException,
      RemoteException, ServiceException {
    try {
      ////////////////////////////////////////////////////////////
      // Connect to Oracle API Catalog
      ////////////////////////////////////////////////////////////
      URL lURL = null;
      lURL = new URL(pArgs[0]);
      FlashlineRegistry repository = new FlashlineRegistryServiceLocator()
          .getFlashlineRegistry(lURL);
      ////////////////////////////////////////////////////////////
      // Login to OER
      ////////////////////////////////////////////////////////////
      AuthToken authToken = repository.authTokenCreate(pArgs[1],
          pArgs[2]);
      ////////////////////////////////////////////////////////////
      // create a criteria object searching for all assets with a
      // custom-data element for test-frequency equal to 'DAILY'
      ////////////////////////////////////////////////////////////
      SearchTerm[] searchTermArray = new SearchTerm[1];
      SearchTerm term = new SearchTerm();
      term.setKey("/asset/custom-data/test-frequency");
      term.setValue("DAILY");
      searchTermArray[0] = term;
      AssetCriteria criteria = new AssetCriteria();
      criteria.setSearchTerms(searchTermArray);
```

```
      ////////////////////////////////////////////////////////
      // perform search, getting back summary objects. loop through
      // objects and perform an action on each one
      ////////////////////////////////////////////////////////
      AssetSummary[] assets = repository.assetQuerySummary(authToken,
          criteria);
      ////////////////////////////////////////////////////////
      // Loop through search results
      ////////////////////////////////////////////////////////
      for (int i = 0; i < assets.length; i++) {
        long assetID = assets[i].getID();
        String testResult = null;
        ////////////////////////////////////////////////////////
        // Update value in the asset
        ////////////////////////////////////////////////////////
        repository.assetUpdateCustomDataNode(
            authToken, assetID, "/asset/custom-data/test-result", testResult);
      }
    } catch (OpenAPIException lEx) {
      System.out.println("ServerCode = " + lEx.getServerErrorCode());
      System.out.println("Message    = " + lEx.getMessage());
      System.out.println("StackTrace:");
      lEx.printStackTrace();
    } catch (RemoteException lEx) {
      lEx.printStackTrace();
    } catch (ServiceException lEx) {
      lEx.printStackTrace();
    } catch (MalformedURLException lEx) {
      lEx.printStackTrace();
    }
  }
}
```

5.4.1.2.12 Use Case: Reading an Asset's Tabs Description

Read the tabs of an asset.

Sample Code

Example 5–16 Use Case: Reading an Asset's Tabs

```
package com.flashline.sample.assetapi;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import com.flashline.registry.openapi.base.OpenAPIException;
import com.flashline.registry.openapi.entity.AuthToken;
import com.flashline.registry.openapi.entity.TabBean;
import com.flashline.registry.openapi.service.v300.FlashlineRegistry;
import
 com.flashline.registry.openapi.service.v300.FlashlineRegistryServiceLocator;
public class AssetReadTabs {
  public static void main(String pArgs[]) throws OpenAPIException,
 RemoteException,
      ServiceException {
    try {
      ////////////////////////////////////////////////////////
      // Connect to Oracle API Catalog
      ////////////////////////////////////////////////////////
```

```
      URL lURL = null;
      lURL = new URL(pArgs[0]);
      FlashlineRegistry repository = new
 FlashlineRegistryServiceLocator().getFlashlineRegistry(lURL);
      TabBean[] lTabBeans = null;
      ////////////////////////////////////
      // Login to OER
      ////////////////////////////////////
      AuthToken authToken = repository.authTokenCreate(pArgs[1],pArgs[2]);
      ////////////////////////////////////
      // read an asset's tabs
      ////////////////////////////////////
      lTabBeans = repository.assetTabsRead(authToken, 559);
    } catch (OpenAPIException lEx) {
      System.out.println("ServerCode = " + lEx.getServerErrorCode());
      System.out.println("Message    = " + lEx.getMessage());
      System.out.println("StackTrace:");
      lEx.printStackTrace();
    } catch (RemoteException lEx) {
      lEx.printStackTrace();
    } catch (ServiceException lEx) {
      lEx.printStackTrace();
    } catch (MalformedURLException lEx) {
      lEx.printStackTrace();
    }
  }
}
```

5.4.1.2.13 Use Case: Retrieve An Asset's Tab Based on TabType Description

Get a specific asset tab by tabtype.

Sample Code

Example 5–17 Use Case: Retrieving an Asset's Tab Based on Tab Type

```
package com.flashline.sample.assetapi;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import com.flashline.registry.openapi.base.OpenAPIException;
import com.flashline.registry.openapi.entity.Asset;
import com.flashline.registry.openapi.entity.AuthToken;
import com.flashline.registry.openapi.entity.TabBean;
import com.flashline.registry.openapi.service.v300.FlashlineRegistry;
import
 com.flashline.registry.openapi.service.v300.FlashlineRegistryServiceLocator;
public class AssetGetTabByType {
  public static void main(String pArgs[]) throws OpenAPIException,
 RemoteException,
      ServiceException {
    try {
      ////////////////////////////////////////////////////////
      // Connect to Oracle API Catalog
      ////////////////////////////////////////////////////////
      URL lURL = null;
      lURL = new URL(pArgs[0]);
      FlashlineRegistry repository = new
 FlashlineRegistryServiceLocator().getFlashlineRegistry(lURL);
```

```
      Asset lAsset = null;
      TabBean lTabBean = null;
      ////////////////////////////////////
      // Login to OER
      ////////////////////////////////////
      AuthToken authToken = repository.authTokenCreate(pArgs[1],pArgs[2]);
      ////////////////////////////////////
      // read an asset
      ////////////////////////////////////
      lAsset = repository.assetRead(authToken, 559);
      ////////////////////////////////////
      // get an asset's tab by tabbeantype
      ////////////////////////////////////
      lTabBean = repository.assetTabRead(authToken, lAsset.getID(), 458);
    } catch (OpenAPIException lEx) {
      System.out.println("ServerCode = " + lEx.getServerErrorCode());
      System.out.println("Message    = " + lEx.getMessage());
      System.out.println("StackTrace:");
      lEx.printStackTrace();
    } catch (RemoteException lEx) {
      lEx.printStackTrace();
    } catch (ServiceException lEx) {
      lEx.printStackTrace();
    } catch (MalformedURLException lEx) {
      lEx.printStackTrace();
    }
  }
}
```

5.4.1.2.14 Use Case: Approving and Unapproving a Tab Description

Approve or unapprove an asset's tab.

Sample Code

Example 5–18 Use Case: Approving and Unapproving a Tab

```
package com.flashline.sample.assetapi;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import com.flashline.registry.openapi.base.OpenAPIException;
import com.flashline.registry.openapi.entity.AuthToken;
import com.flashline.registry.openapi.service.v300.FlashlineRegistry;
import
 com.flashline.registry.openapi.service.v300.FlashlineRegistryServiceLocator;

public class ApproveUnapproveTab {
  public static void main(String pArgs[]) throws OpenAPIException,
 RemoteException,
      ServiceException {
    try {
      ////////////////////////////////////////////////////////
      // Connect to Oracle API Catalog
      ////////////////////////////////////////////////////////
      URL lURL = null;
      lURL = new URL(pArgs[0]);
      FlashlineRegistry repository = new
 FlashlineRegistryServiceLocator().getFlashlineRegistry(lURL);
```

```
      ////////////////////////////////
      // Login to OER
      ////////////////////////////////
      AuthToken authToken = repository.authTokenCreate(pArgs[1],pArgs[2]);
      ////////////////////////////////
      // approve an asset tab
      ////////////////////////////////
      repository.assetTabApprove(authToken, 559, 1864);
      ////////////////////////////////
      // unapprove an asset tab
      ////////////////////////////////
      repository.assetTabUnapprove(authToken, 559, 1864);
    } catch (OpenAPIException lEx) {
      System.out.println("ServerCode = " + lEx.getServerErrorCode());
      System.out.println("Message    = " + lEx.getMessage());
      System.out.println("StackTrace:");
      lEx.printStackTrace();
    } catch (RemoteException lEx) {
      lEx.printStackTrace();
    } catch (ServiceException lEx) {
      lEx.printStackTrace();
    } catch (MalformedURLException lEx) {
      lEx.printStackTrace();
    }
  }
}
```

5.4.1.2.15 Use Case: Reading an Asset's Metadata for a Given Tab Description

Read the metadata of an asset based on the given tab.

Sample Code

Example 5–19 Use Case: Reading an Asset's Metadata for a Given Tab

```
package com.flashline.sample.assetapi;

import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;

import javax.xml.rpc.ServiceException;

import com.flashline.registry.openapi.base.OpenAPIException;
import com.flashline.registry.openapi.entity.Asset;
import com.flashline.registry.openapi.entity.AssetMetadataElement;
import com.flashline.registry.openapi.entity.AssetMetadataTableElement;
import com.flashline.registry.openapi.entity.AuthToken;
import com.flashline.registry.openapi.service.v300.FlashlineRegistry;
import
com.flashline.registry.openapi.service.v300.FlashlineRegistryServiceLocator;

public class AssetGetMetadataByTab {
  public static void main(String pArgs[]) throws OpenAPIException,
RemoteException,
  ServiceException {
    try {

      ////////////////////////////////////////////////////////
      // Connect to Oracle API Catalog
```

```
      ////////////////////////////////////////////////////////
      URL lURL = null;
      lURL = new URL(pArgs[0]);
      FlashlineRegistry repository = new
FlashlineRegistryServiceLocator().getFlashlineRegistry(lURL);
      Asset lAsset = null;

      ////////////////////////////////////
      // Login to OER
      ////////////////////////////////////
      AuthToken authToken = repository.authTokenCreate(pArgs[1],pArgs[2]);

      ////////////////////////////////////
      // read an asset
      ////////////////////////////////////
      lAsset = repository.assetRead(authToken, 589);

      ////////////////////////////////////
      // Get the metadata elements based
      // on the asset ID and the tab name
      ////////////////////////////////////
      AssetMetadataElement[] lElements =
repository.assetReadTabMetadata(authToken, lAsset.getID(), "Overview");

      //An AssetMetadataElement represents a custom or mandatory data element of
an asset
      for (AssetMetadataElement lElement : lElements) {
        //This represents a TABLE element type
        if (lElement.getValue() instanceof AssetMetadataTableElement) {
          AssetMetadataTableElement lTable =
(AssetMetadataTableElement)lElement.getValue();
          System.out.println(lElement.getDisplayName());
          //A TABLE can have multiple elements
          for (AssetMetadataElement lTableElement : lTable.getElements()) {
            //An element of a TABLE can be another TABLE
            if (lTableElement.getValue() instanceofAssetMetadataTableElement) {

System.out.println(((AssetMetadataTableElement)lTableElement.getValue()).getDispla
yName());
              for (AssetMetadataElement lChildElement :
((AssetMetadataTableElement)lTableElement.getValue()).getElements()) {
                System.out.println(lChildElement.getDisplayName() + " : " +
lChildElement.getValue());
              }
            } else {
              //Or an element of a TABLE can be a regular value
              System.out.println(lTableElement.getDisplayName() + " : " +
lTableElement.getValue());
            }
          }
          //This represents a MULTIPLE ITEM LIST
        }  else if (lElement.getValue() instanceof String[]) {
          System.out.println(lElement.getDisplayName());
          for (String lString : (String[])lElement.getValue()) {
            System.out.println(lString);
          }
          //This represents a SINGLE ITEM
        } else {
          System.out.println(lElement.getDisplayName() + " : " +
lElement.getValue());
```

```
        }
      }
    } catch (OpenAPIException lEx) {
      System.out.println("ServerCode = " + lEx.getServerErrorCode());
      System.out.println("Message    = " + lEx.getMessage());
      System.out.println("StackTrace:");
      lEx.printStackTrace();
    } catch (RemoteException lEx) {
      lEx.printStackTrace();
    } catch (ServiceException lEx) {
      lEx.printStackTrace();
    } catch (MalformedURLException lEx) {
      lEx.printStackTrace();
    }
  }
}
```

5.4.2 Department API

This section provides a use case for the Department API that describes how to create, update, query, and delete departments in Oracle API Catalog.

5.4.2.1 Overview

Departments can be created, read, queried for, and modified. These operations are described below. Bear in mind that after a Department is created, it cannot be deleted. Only two Department attributes are meaningful to a user: name and description.

Additional Import(s) Required

```
import com.flashline.registry.openapi.entity.Department;
```

5.4.2.2 Use Case: Manipulate Departments

Description

The following sample code illustrates typical tasks involving the manipulation of departments in Oracle API Catalog. This includes creation, updating, querying, and deleting.

Sample Code

Example 5–20 Use Case: Manipulate Departments

```
package com.flashline.sample.departmentapi;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import java.util.Calendar;
import javax.xml.rpc.ServiceException;
import com.flashline.registry.openapi.base.OpenAPIException;
import com.flashline.registry.openapi.entity.AuthToken;
import com.flashline.registry.openapi.entity.Department;
import com.flashline.registry.openapi.query.DepartmentCriteria;
import com.flashline.registry.openapi.service.v300.FlashlineRegistry;
import
 com.flashline.registry.openapi.service.v300.FlashlineRegistryServiceLocator;
public class Departments {
  public static void main(String pArgs[]) throws java.rmi.RemoteException,
```

```
      OpenAPIException {
    try {
      ////////////////////////////////////////////////////////
      // Connect to Oracle API Catalog
      ////////////////////////////////////////////////////////
      URL lURL = null;
      lURL = new URL(pArgs[0]);
      FlashlineRegistry repository = new FlashlineRegistryServiceLocator()
          .getFlashlineRegistry(lURL);
      // ////////////////////////////////
      // Authenticate with OER
      // ////////////////////////////////
      AuthToken authToken = repository.authTokenCreate(pArgs[1],
          pArgs[2]);
      // ////////////////////////////////
      // Create a new department
      // Each Department requires a unique name. Descriptions are optional.
      // ////////////////////////////////
      Department dept = repository.departmentCreate(authToken,
          "My Dept "+Calendar.getInstance().getTimeInMillis(), "A New
Department");
      // ////////////////////////////////
      // Read a department
      // To read a Department you must have the Department name.
      // ////////////////////////////////
      Department dept2 = repository.departmentRead(authToken,
          "ADepartment");
      // ////////////////////////////////
      // Query for a department
      //
      // To query for a Department you must fill out a
      // DepartmentCriteria object with an array of SearchTerms. A SearchTerm
      // is a key/value pair. Currently the only valid key is "name".
      //
      // A query for name is a match if the value for the name term
      // occurs anywhere in the name of the department. For example,
      // a search for fred matches fred, alfred, and fredrick.
      // ////////////////////////////////
      DepartmentCriteria criteria = new DepartmentCriteria();
      criteria.setNameCriteria("DepartmentName");
      Department[] depts = repository.departmentQuery(authToken,
          criteria);
      // ////////////////////////////////
      // Update a department
      //
      // To update a Department you need only to modify a Department
      // reference and call departmentUpdate...
      // ////////////////////////////////
      String lOldName = dept.getName();
      String lNewName = "New " + dept.getName();
      Department dept3 = repository.departmentRead(authToken, lOldName);
      dept3.setName(lNewName);
      repository.departmentUpdate(authToken, dept3);
    } catch (OpenAPIException lEx) {
      System.out.println("ServerCode = " + lEx.getServerErrorCode());
      System.out.println("Message    = " + lEx.getMessage());
      System.out.println("StackTrace:");
      lEx.printStackTrace();
    } catch (RemoteException lEx) {
      lEx.printStackTrace();
```

```
    } catch (ServiceException lEx) {
      lEx.printStackTrace();
    } catch (MalformedURLException lEx) {
      lEx.printStackTrace();
    }
  }
}
```

5.4.3 Localization of REX Clients

This section provides use cases for the localization of REX clients that describe how to create localized messages from REX Exceptions and REX Audit Messages in Oracle API Catalog.

5.4.3.1 Overview

Statuses are passed back to a REX client as either an OpenAPIException or AuditMsg object. OpenAPIException objects are used for exceptions, whereas, AuditMsg objects are used for processes that run asynchronously. Both of these objects return a text error message to the REX client.

The interface of both objects has been expanded to include an error code and a list of message arguments so that REX clients can display error or status messages in another language. Clients can continue to use the standard error messages or they can ignore the message and use the error code and the message arguments to construct their own error message.

For example, if you want to localize an application that uses REX, you would first get the properties file listing all the possible error messages. The messages look something like this:

```
ERR_9008 = Error updating project with ID = [{0}].
```

Then you must translate all the messages as necessary:

```
ERR_9008 = Errorway updatingay ojectpray ithway IDway = [{0}].
```

If the client code tries to modify a project with ID=123, and that modification fails, then your end-users get an exception with this error message:

```
Error updating project with ID = [123].
```

If you want to display that error in a local language (such as, Pig Latin), you would take the error code, 9008, and look it up in your translated file to get the string "Errorway updatingay ojectpray ithway IDway = [{0}]." Then you would use the message arguments to replace the tokens. In this case, there is only one string, "123", so you should be able to find one message argument.

You can then construct a custom error message for your end-users:

```
Errorway updatingay ojectpray ithway IDway = [123].
```

5.4.3.2 Use Case: Creating Localized Messages from REX Exceptions

Description

- From the OpenAPIException get the server error code and the message arguments

- Get the resource bundle for the OpenAPIExceptions appropriate for the client locale
- Get the string associated with the error code and replace the token with the message arguments

Sample Code

Example 5–21 Use Case: Creating Localized Messages from REX Exceptions

```
package com.flashline.sample.localization;
import java.net.URL;
import java.text.MessageFormat;
import java.util.ResourceBundle;
import com.flashline.registry.openapi.base.OpenAPIException;
import com.flashline.registry.openapi.entity.AuthToken;
import com.flashline.registry.openapi.entity.Project;
import com.flashline.registry.openapi.service.v300.FlashlineRegistry;
import
 com.flashline.registry.openapi.service.v300.FlashlineRegistryServiceLocator;
public class SyncTest {
    private static final int INVALID_PROJECT_ID = 8672609;
    public static void main(String[] args) throws Exception {
      URL lURL = new
 URL("http://localhost:9080/registry/services/FlashlineRegistry");
      FlashlineRegistry reg = new
 FlashlineRegistryServiceLocator().getFlashlineRegistry(lURL);
      AuthToken token = reg.authTokenCreate("admin", "n0pa55w0rd");
      try {
        Project project = reg.projectRead(token,INVALID_PROJECT_ID);
      } catch (OpenAPIException ex) {
        String msg =
 createMessage(ex.getServerErrorCode(),ex.getMessageArguments());
        System.out.println(msg);
      }
    }
    private static String createMessage(int pServerErrorCode, Object[] pArgs) {
      ResourceBundle mResourceBundle =
 ResourceBundle.getBundle("com.flashline.sample.localization.sync_error
_messages");
      return MessageFormat.format(mResourceBundle.getString("ERR
_"+pServerErrorCode), pArgs);
    }
}
```

5.4.3.3 Use Case: Creating Localized Messages from REX Audit Messages

Description

- From the AuditMsg and the ImpExpJob get the server error code and the message arguments from the AuditMsg
- Get the resource bundle for audit messages appropriate for the client locale
- Get the string associated with the error code and replace the token with the message arguments

Sample Code

Example 5–22 Use Case: Creating Localized Messages from REX Audit Messages

```
package com.flashline.sample.localization;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import java.text.MessageFormat;
import java.util.ResourceBundle;
import javax.activation.DataHandler;
import javax.xml.rpc.ServiceException;
import org.apache.axis.client.Stub;
import org.apache.soap.util.mime.ByteArrayDataSource;
import com.flashline.registry.openapi.base.OpenAPIException;
import com.flashline.registry.openapi.entity.AuthToken;
import com.flashline.registry.openapi.entity.ImpExpJob;
import com.flashline.registry.openapi.service.v300.FlashlineRegistry;
import
 com.flashline.registry.openapi.service.v300.FlashlineRegistryServiceLocator;
public class AsyncTest {
  public void run() throws MalformedURLException, ServiceException,
 OpenAPIException, RemoteException{
    URL lURL = new
 URL("http://localhost:9080/registry/services/FlashlineRegistry");
    FlashlineRegistry reg = new
 FlashlineRegistryServiceLocator().getFlashlineRegistry(lURL);
    AuthToken token = reg.authTokenCreate("admin", "n0pa55w0rd");
    try {
      File lFile = new
 File("samples/com/flashline/sample/localization/asynctest.zip");
      //Import the file and save to db
      InputStream lIS = new FileInputStream(lFile);
      ByteArrayDataSource lDataSource = new ByteArrayDataSource(lIS,
 "application/x-zip-compressed");
      DataHandler lDH = new DataHandler(lDataSource);
      // add the attachment
      ((Stub)reg).addAttachment(lDH);
      ImpExpJob lJob = reg.importExecute(token, "flashline", null, "Import Assets
 Test", null);
      boolean lPassed = false;
      for(int i=0; i<1000; i++){
        lJob = reg.importStatus(token, lJob);
        System.out.println("Import Job ["+lJob.getID()+"] - State:
 "+lJob.getState());
        String msg =
createMessage(lJob.getAuditMsg().getSummaryID(),lJob.getAuditMsg().getSummaryArgs
());
        System.out.println(msg);
        if( lJob.getState().equals("completed")){
          lPassed = true;
          break;
        }
        try {
```

```
          Thread.sleep(1000);
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
      }
      if (lPassed){
         System.out.println("Import Completed");
      }
    } catch (OpenAPIException ex) {
      String msg =
 createMessage(ex.getServerErrorCode(),ex.getMessageArguments());
      System.out.println(msg);
    } catch (FileNotFoundException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
  }
  /**
   * @param args
   * @throws ServiceException
   * @throws RemoteException
   * @throws MalformedURLException
   * @throws OpenAPIException
   */
  public static void main(String[] args) throws OpenAPIException,
 MalformedURLException, RemoteException, ServiceException {
    AsyncTest test = new AsyncTest();
    test.run();
  }
  private static String createMessage(int pServerErrorCode, Object[] pArgs) {
    ResourceBundle mResourceBundle =
 ResourceBundle.getBundle("com.flashline.sample.localization.async_error
_messages");
    return MessageFormat.format(mResourceBundle.getString("ERR
_"+pServerErrorCode), pArgs);
  }
  private String readZip(String pFileName) throws IOException {
    int    lNumRead = 0;
    char[] lBuf = new char[2048];
    StringBuffer lQuery = new StringBuffer();
    InputStreamReader lReader = new
 InputStreamReader(getClass().getResourceAsStream(pFileName));
    while( (lNumRead=lReader.read(lBuf)) != -1){
      lQuery.append(lBuf, 0, lNumRead);
    }
    return lQuery.toString();
  }
}
```

5.4.4 System Settings API

This section provides a use case for the System Settings API that describes how to query for system settings in Oracle API Catalog.

5.4.4.1 Overview

Within the Oracle API Catalog's System Settings section administrators can configure the basic operations and enable/disable specific features. The System Settings API provides a mechanism to query these system settings.

> **Note:** Users are allowed only to query the system settings for values, the system settings cannot be set or modified through REX.

To query the system settings, the following package import(s) are required:

```
import com.flashline.registry.openapi.entity.SettingValue;
import com.flashline.registry.openapi.entity.AuthToken;
import com.flashline.registry.openapi.query.SystemSettingsCriteria;
```

Reserved Methods

The `systemSettingsAddBundle` method is reserved for future use and is not intended for general use.

5.4.4.2 Use Case: Query for System Settings

Description

Query for system settings in REX.

Sample Code

Example 5–23 Use Case: Query for System Settings

```
package com.flashline.sample.systemsettingsapi;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import com.flashline.registry.openapi.base.OpenAPIException;
import com.flashline.registry.openapi.entity.AuthToken;
import com.flashline.registry.openapi.entity.SettingValue;
import com.flashline.registry.openapi.query.SystemSettingsCriteria;
import com.flashline.registry.openapi.service.v300.FlashlineRegistry;
import
 com.flashline.registry.openapi.service.v300.FlashlineRegistryServiceLocator;
public class SystemSettings {
  public static void main(String pArgs[]) throws OpenAPIException,
 RemoteException,
      ServiceException {
    try {
      ////////////////////////////////////////////////////////
      // Connect to Oracle API Catalog
      ////////////////////////////////////////////////////////
      URL lURL = null;
      lURL = new URL(pArgs[0]);
      FlashlineRegistry repository = new
 FlashlineRegistryServiceLocator().getFlashlineRegistry(lURL);
      // ////////////////////////////
      // Authenticate with OER
      // ////////////////////////////
      AuthToken authToken = repository.authTokenCreate(pArgs[1],pArgs[2]);
```

```
      // ////////////////////////////
      // Set Application Token on AuthToken object.  This is supplied by OER
      //authToken.setApplicationToken("TokenString");
      // ////////////////////////////////////
      //Read all available system settings
      //Create an empty Criteria object.  No criteria returns all settings.
      SystemSettingsCriteria lCriteria = new SystemSettingsCriteria();
      lCriteria.setNameCriteria("enterprise.defaults.displayname.field");
      SettingValue[] lValues = repository.systemSettingsQuery(authToken,
 lCriteria);
      for (int i=0;i<lValues.length;i++) {
        SettingValue lValue = lValues[i];
        System.out.println("Setting Name: " + lValue.getDescriptor().getName());
        System.out.println("Setting Value: " + lValue.getValue());
      }
      // ////////////////////////////////////
      //Read a specific setting
      lCriteria.setNameCriteria("cmee.server.companyname");
      lValues = repository.systemSettingsQuery(authToken, lCriteria);
      for (int i=0;i<lValues.length;i++) {
        SettingValue lValue = lValues[i];
        System.out.println("Setting Name: " + lValue.getDescriptor().getName());
        System.out.println("Setting Value: " + lValue.getValue());
      }
      // ////////////////////////////////////
      //Read a specific section
      lCriteria.setSectionCriteria("general");
      lValues = repository.systemSettingsQuery(authToken, lCriteria);
      for (int i=0;i<lValues.length;i++) {
        SettingValue lValue = lValues[i];
        System.out.println("Setting Name: " + lValue.getDescriptor().getName());
        System.out.println("Setting Value: " + lValue.getValue());
      }
    } catch (OpenAPIException lEx) {
      System.out.println("ServerCode = " + lEx.getServerErrorCode());
      System.out.println("Message    = " + lEx.getMessage());
      System.out.println("StackTrace:");
      lEx.printStackTrace();
    } catch (RemoteException lEx) {
      lEx.printStackTrace();
    } catch (ServiceException lEx) {
      lEx.printStackTrace();
    } catch (MalformedURLException lEx) {
      lEx.printStackTrace();
    }
  }
}
```

5.4.5 User API

This section provides a use case for the User API that describes how to create, retrieve, update, and deactivate users or query for users.

5.4.5.1 Overview

The User Subsystem provides a web services-based mechanism that is used to create, read, update, query, and otherwise manipulate Oracle API Catalog User accounts.

Additional Import(s) Required

```
import com.flashline.registry.openapi.entity.RegistryUser;
import com.flashline.registry.openapi.query.UserCriteria;
```

5.4.5.2 Use Case: Manipulating Users

Description

- Create a new user.
- Retrieve an existing user.
- Update a user.
- Deactivate a user.
- Query for users.

Sample Code

Example 5–24 Use Case: Manipulate User

```
package com.flashline.sample.userapi;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import java.util.Calendar;
import javax.xml.rpc.ServiceException;
import com.flashline.registry.openapi.base.OpenAPIException;
import com.flashline.registry.openapi.entity.AuthToken;
import com.flashline.registry.openapi.entity.RegistryUser;
import com.flashline.registry.openapi.query.UserCriteria;
import com.flashline.registry.openapi.service.v300.FlashlineRegistry;
import
 com.flashline.registry.openapi.service.v300.FlashlineRegistryServiceLocator;
public class Users {
  public static void main(String pArgs[]) throws OpenAPIException,
 RemoteException,
      ServiceException {
    try {
      ////////////////////////////////////////////////////////
      // Connect to Oracle API Catalog
      ////////////////////////////////////////////////////////
      URL lURL = null;
      lURL = new URL(pArgs[0]);
      FlashlineRegistry repository = new FlashlineRegistryServiceLocator()
          .getFlashlineRegistry(lURL);
      // ////////////////////////////
      // Authenticate with OER
      // ////////////////////////////
      AuthToken authToken = repository.authTokenCreate(pArgs[1],
          pArgs[2]);
      // ---------------------------------------
      // Create a new user
      String lUserName = "testUserCreate
_"+Calendar.getInstance().getTimeInMillis();
      String lFirstName = "testUserCreate_FirstName";
      String lLastName = "testUserCreate_LastName";
      String lEmail = lUserName+"@example.com";
      String lPassword = "testUserCreate_Password";
```

```
        boolean lMustChangePassword = false;
        boolean lPasswordNeverExpires = false;
        boolean lAssignDefafultRoles = true;
        RegistryUser RbacRegistrySecUser = repository.userCreate(
            authToken, lUserName, lFirstName, lLastName, lEmail, lPassword,
            lMustChangePassword, lPasswordNeverExpires, lAssignDefafultRoles);
        // ----------------------------------------
        // Read a User
        long lId = 50000; // user id must exist in OER
        RegistryUser lUser1 = repository.userRead(authToken,
            lId);
        // ----------------------------------------
        // Update a User
        lUser1.setActiveStatus(10);
        lUser1.setUserName("xxx");
        lUser1.setPhoneNumber("412-521-4914");
        lUser1.setMustChangePassword(true);
        lUser1.setPasswordNeverExpires(false);
        lUser1.setPassword("changed_password");
        lUser1.setEmail("newaddress@bea.com");
        try {
          lUser1 = repository.userUpdate(authToken,
              lUser1);
        } catch (OpenAPIException e) {
          e.printStackTrace();
        }
        // ----------------------------------------
        // Deactivate a User
        RegistryUser lUser2 = null;
        try {
          lUser2 = repository.userDeactivate(authToken, lId);
        } catch (OpenAPIException e) {
          e.printStackTrace();
        }
        // ----------------------------------------
        // Query for Users
        RegistryUser lUsers[] = null;
        UserCriteria lUserCriteria = null;
        lUserCriteria = new UserCriteria();
        lUserCriteria.setNameCriteria("testname");
        lUsers = repository.userQuery(authToken,
            lUserCriteria);
      } catch (OpenAPIException lEx) {
        System.out.println("ServerCode = " + lEx.getServerErrorCode());
        System.out.println("Message    = " + lEx.getMessage());
        System.out.println("StackTrace:");
        lEx.printStackTrace();
      } catch (RemoteException lEx) {
        lEx.printStackTrace();
      } catch (ServiceException lEx) {
        lEx.printStackTrace();
      } catch (MalformedURLException lEx) {
        lEx.printStackTrace();
      }
    }
  }
```