Oracle® Fusion Middleware

Developing and Customizing Applications for Oracle Identity Manager

11*g* Release 2 (11.1.2.3.0)

E56655-13

January 2019

ORACLE®

Oracle Fusion Middleware Developing and Customizing Applications for Oracle Identity Manager, 11*g* Release 2 (11.1.2.3.0)

E56655-13

Primary Author: Anju Poovaiah

Contributing Author: Debapriya Datta

Contributor: Sanjay Rallapalli

Contents

List of Figures

List of Tables

Preface

The *Oracle Fusion Middleware Developer's Guide for Oracle Identity Manager* describes how to develop and customize various components and features of Oracle Identity Manager.

Audience

This guide is intended for developers who use Oracle Identity Manager development tools to customize the product according to the requirements of an organization. The customization involves using APIs, configuring requests and approval workflows, developing connectors by using Identity Connector Framework, Generic Technology Connector, or Adapter Factory, and customizing the user interface.

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, refer to the following documents:

- *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager*
- *Oracle Fusion Middleware User's Guide for Oracle Identity Manager*
- *Oracle Fusion Middleware Installation Guide for Oracle Identity and Access Management*
- *Oracle Fusion Middleware Enterprise Deployment Guide for Oracle Identity Management*
- *Oracle Fusion Middleware High Availability Guide*
- *Oracle Fusion Middleware Administrator's Guide for Oracle Access Management*
- *Oracle Fusion Middleware Administrator's Guide for Oracle Adaptive Access Manager*
- *Oracle Fusion Middleware Developer's Guide for Oracle Adaptive Access Manager*

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.

What's New In This Guide

This preface provides a summary of new features and updates to *Developing and Customizing Applications for Oracle Identity Manager* 11*g* Release 2 (11.1.2.3.0).

Follow the pointers into this guide to get more information about the features and how to use them. This document is the new edition of the formerly titled *Oracle Fusion Middleware Developer's Guide for Oracle Identity Manager*.

Updates in January 2019 Documentation Refresh for 11*g* Release 2 (11.1.2.3)

This revision of *Developing and Customizing Applications for Oracle Identity Manager* contains bug fixes and editorial corrections.

Updates in July 2018 Documentation Refresh for 11*g* Release 2 (11.1.2.3)

This revision of *Developing and Customizing Applications for Oracle Identity Manager* contains bug fixes and editorial corrections.

Updates in April 2018 Documentation Refresh for 11*g* Release 2 (11.1.2.3)

This revision of *Developing and Customizing Applications for Oracle Identity Manager* contains bug fixes and editorial corrections.

Updates in January 2018 Documentation Refresh for 11*g* Release 2 (11.1.2.3)

This revision of *Developing and Customizing Applications for Oracle Identity Manager* includes the following update:

- Oracle Identity Manager provides TLSv1.2 protocol support for SSL communication between Java Connector Server and Oracle Identity Manager. See Chapter 5, "Developing Identity Connectors Using Java".

Updates in December 2017 Documentation Refresh for 11*g* Release 2 (11.1.2.3)

This revision of *Developing and Customizing Applications for Oracle Identity Manager* includes the following update:

- Oracle Identity Manager provides TLSv1.2 protocol support for SSL communication between .Net Connector Server and Oracle Identity Manager. See

Section 4.6.2, "Using the .NET Connector Server".

Updates in October 2017 Documentation Refresh for 11*g* Release 2 (11.1.2.3)

This revision of *Developing and Customizing Applications for Oracle Identity Manager* contains bug fixes and editorial corrections.

Updates in January 2017 Documentation Refresh for 11*g* Release 2 (11.1.2.3)

This revision of *Developing and Customizing Applications for Oracle Identity Manager* contains bug fixes and editorial corrections.

Updates in October 2016 Documentation Refresh for 11*g* Release 2 (11.1.2.3)

This revision of *Developing and Customizing Applications for Oracle Identity Manager* contains the following updates:

- SCIM resources are secured by custom Oracle Web Services Manager (OWSM) policy. See Section 21.6, "Securing SCIM Resources".
- After you apply bundle patch 11.1.2.3.161018, Oracle Identity Manager provides a JSON Web Token (JWT) service to simplify the use of Oracle Identity Manager SCIM-REST service. See Chapter 22, "Using the JSON Web Token Service".

This revision also contains bug fixes and editorial corrections.

Updates in July 2016 Documentation Refresh for 11*g* Release 2 (11.1.2.3)

This revision of *Developing and Customizing Applications for Oracle Identity Manager* contains bug fixes and editorial corrections.

Updates in January 2016 Documentation Refresh for 11*g* Release 2 (11.1.2.3)

This revision of *Developing and Customizing Applications for Oracle Identity Manager* contains bug fixes and editorial corrections.

Updates in August 2015 Documentation Refresh for 11*g* Release 2 (11.1.2.3.0)

This revision of *Developing and Customizing Applications for Oracle Identity Manager* contains bug fixes and editorial corrections.

Updates in June 2015 Documentation Refresh for 11*g* Release 2 (11.1.2.3.0)

This revision of *Developing and Customizing Applications for Oracle Identity Manager* contains bug fixes and editorial corrections.

New and Changed Features for 11*g* Release 2 (11.1.2.3.0)

Oracle Identity Manager 11*g* Release 2 (11.1.2.3.0) includes the following new and changed development features for this document.

- Identity REST services that are based on the System for Cross-Domain Identity Management (SCIM) protocol, which provides functionality for self-service, user, role/group, organization, and password policy management. See Chapter 21, "Using SCIM/REST Services".

Other Significant Changes in this Document for 11*g* Release 2 (11.1.2.3.0)

For 11*g* Release 2 (11.1.2.3.0), this guide has been updated in several ways. Following are the sections that have been added or changed.

- Removed information about Oracle Identity Manager overview and architecture.
- Removed information about security architecture in Oracle Identity Manager as the authorization and security model in Oracle Identity Manager 11*g* Release 2 (11.1.2.3.0) works on the basis of the custom admin role assignment to a user. See "Managing Admin Roles" section in *Performing Self Service Tasks with Oracle Identity Manager*.
- Moved information about creating and managing Generic Technology Connectors to *Administering Oracle Identity Manager*. See "Managing Generic Connectors" section in that book.
- Removed information about Segregation of Duties (SoD) as audit violations and SoD is managed in this release by using the Identity Audit feature. See "Managing Identity Audit" in *Performing Self Service Tasks with Oracle Identity Manager*.

  However, SoD check with Oracle Application Access Controls Governor (OAACG) is supported for backward compatibility. For information about SoD check with OAACG 8.6.5, refer to the following URL:

  `https://docs.oracle.com/cd/E40329_01/dev.1112/e27150/segduties.htm#OMDEV3109`
- Revised procedures for customizing the interface, as a result of the new UI in Oracle Identity Self Service. See Chapter 19, "Customizing the Interface".
- Removed information about using SPML services as SPML is not supported in this release.
- Removed information about the callback service as it is not supported in this release.
- Removed information about Oracle Identity Manager customization by using the Design Console as various Design Console operations have been deprecated in this release.

Part I

Application Provisioning

This part describes how to configure application-specific connectors.

It contains the following chapters:

1 Developing Application Instances

An application instance is a provisionable entity. It is a combination of IT resource instance (target connectivity and connector configuration) and resource object (provisioning mechanism). Application instances have business-friendly names that are easier to remember. Creating and managing application instances are performed by using the Application Instance section of Oracle Identity System Administration.

Application instances can be connected or disconnected. A connected application instance has a connector defined for the provisioning of entities. A disconnected application instance is used for the provisioning of a disconnected resource, for which a connector is not defined, and therefore, the provisioning is performed manually by the administrator.

For information about application instance concepts and how to create and manage application instances, see "Managing Application Instances" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager*.

This chapter describes how application developers can manage resource objects, which is a component of application instance, by using the Design Console. In addition, it describes the procedure to convert a disconnected application instance to a connected application instance. For information about creating and managing IT resources, see "Managing IT Resources" in *Administering Oracle Identity Manager*.

This chapter includes the following topics:

- Managing Resources By Using the Design Console
- Converting a Disconnected Application Instance to Connected Application Instance

1.1 Managing Resources By Using the Design Console

This chapter describes resource management in the Design Console. It contains the following sections:

Note: Only the users belonging to the SYSTEM ADMINISTRATORS group of Oracle Identity Manager can log in to Design Console.

- Overview of Resource Management
- IT Resources Type Definition Form

1.1.1 Overview of Resource Management

The Resource Management folder provides you with tools to manage Oracle Identity Manager resources. This folder contains the following forms:

- **IT Resources Type Definition**: Use this form to create resource types that are displayed as lookup values on the IT Resources form.
- **Rule Designer**: Use this form to create rules that can be applied to password policy selection, automatic role membership, provisioning process selection, task assignment, and prepopulating adapters.
- **Resource Objects**: Use this form to create and manage resource objects. These objects represent resources that you want to make available to users and organizations.

See Also: See Chapter 3, "Using the Adapter Factory" for more information about adapters and adapter tasks

1.1.2 IT Resources Type Definition Form

The IT Resources Type Definition form is in the Resource Management folder. You use the IT Resources Type Definition form to classify IT resource types, for example, AD, Microsoft Exchange, and Solaris. Oracle Identity Manager associates resource types with resource objects that it provisions to users and organizations.

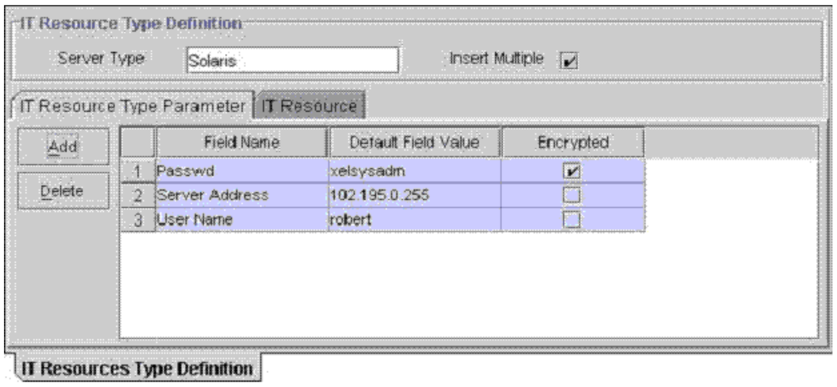
After you define an IT resource type on this form, it is available for selection when you define an IT resource. The type is displayed in the Create IT Resource and Manage IT Resource pages of Advanced Administration.

IT resource types are templates for the IT resource definitions that reference them. If an IT resource definition references an IT resource type, the resource inherits all of the parameters and values in the IT resource type. The IT resource type is the general IT classification, for example, Solaris. The resource is an instance of the type, for example, Solaris for Statewide Investments.

You must associate every IT resource definition with an IT resource type.

Figure 1–1 shows the IT Resources Type Definition form.

Figure 1–1 The IT Resources Type Definition Form


IT Resource Type Definition
IT Resource Type Parameter
IT Resource
Add
Delete
IT Resources Type Definition

Server Type: Solaris
Insert Multiple: ✓

	Field Name	Default Field Value	Encrypted
1	Passwd	xelsysadm	✓
2	Server Address	102.195.0.255	☐
3	User Name	robert	☐

Table 1–1 describes the fields of the IT Resources Type Definition form.

Table 1–1 Fields of the IT Resources Type Definition Form

Field Name	Description
Server Type	The name of the IT resource type
Insert Multiple	Specifies whether or not this IT resource type can be referenced by more than one IT resource

1.1.2.1 Defining a Template (a Resource Type) for IT Resources

To define an IT resource type:

1. Enter the name of the IT resource type in the **Server Type** field, for example, Solaris.
2. To make the IT resource type available for multiple IT resources, select **Insert Multiple**.
3. Click **Save**.

   The IT resource type is defined. You can select it when defining IT resources in the Create IT Resource page of Advanced Administration.

1.2 Converting a Disconnected Application Instance to Connected Application Instance

To describe the procedure to convert a disconnected application instance to a connected application instance, the following assumptions have been made:

- A disconnected application instance exists in Oracle Identity Manager deployment, for example, the production environment. This disconnected application instance will be exported to another deployment of Oracle Identity Manager, for example, a test environment, and converted to a connected application instance. After testing the connected application instance in the test environment, it will be imported in the production environment again.

  Note: Optionally, the disconnected resource can be converted to a connected resource in the same environment. See "Modifying the Application Instance from Disconnected to Connected" on page 1-6 for further details.

- The application instance, process definition, forms, IT resource type definition, and IT resource retain the same name while converting a disconnected application instance to connected application instance.

The following are the broad-level steps to convert a disconnected application instance to a connected application instance:

- Import the existing disconnected resource from the existing environment to the test environment.
- Modify the implementation of the application instance, such as resource object definition and process definition.
- Test the application instance by provisioning it to users and validating the behavior for enable, disable, revoke, and update tasks.

- Export the new connected resource from the test environment and import it to the production environment.

Note:

- Only the resource is exported between environments and not the application instance.
- This section outlines the steps to import/export the resource of the application instance by using the Deployment Manager. Alternatively, the connector upgrade utility can also be used for import/export of the resource. See "Managing Connector Lifecycle" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager* for information about using the connector upgrade utility.

1.2.1 Creating a Disconnected Application Instance in the Production Environment

To create a disconnected application instance in the production environment:

1. Login to Oracle Identity System Administration.
2. Click **Sandboxes** to access sandbox management, create a sandbox, and activate it. See "Managing Sandboxes" on page 19-1 for information about sandboxes and how to create, activate, and publish sandboxes.
3. Under Configuration, click **Application Instances**. Click **Create** on the toolbar to open the Create Application Instance page.
4. Enter values in the Name and Display Name fields, such as `LaptopApplicationInstance`.
5. Select the **Disconnected** option to specify a disconnected application instance. Selecting the Disconnected option disables the Resource Object and IT Resource Instance fields in the page.
6. Click **Save**, and then click **OK** to confirm creation of the FinApp application instance. The artifacts for a disconnected application instance are created.
7. Go to the Manage Sandboxes page, and publish the sandbox.

Upon successful creation of the application instance, organization and entitlements can be configured if necessary. For testing purpose, create four or five users and provision the newly created disconnected application instance to the users. Ensure that the users have the application instance in one of the following status: Provisioned, Enabled, Disabled, and Revoke. Try modifying one of the users to ensure that the account can be successfully updated.

1.2.2 Exporting Disconnected Application Instance From Test Environment

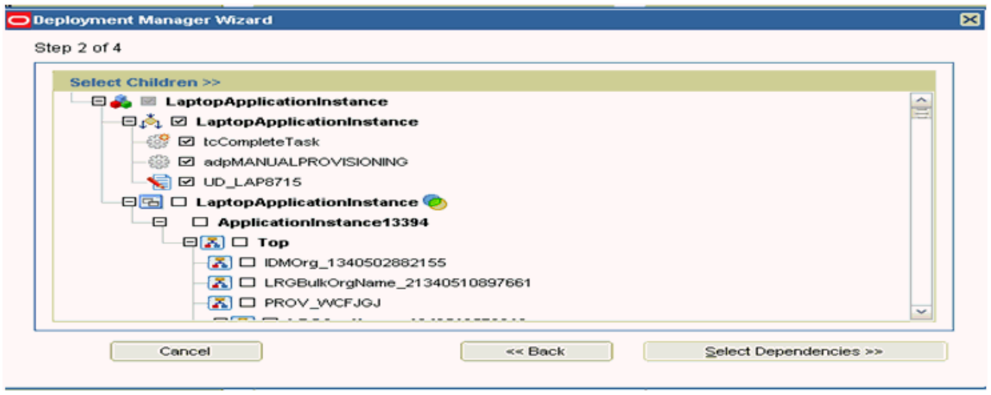
To export the disconnected application instance from the test environment:

1. Login to Oracle Identity System Administration. In the left pane, under System Management, click **Export**. The Deployment Manager wizard is displayed in a new window.
2. Search for the disconnected application instance. To do so, in the search section, select Resource from the list, enter the name of the disconnected application instance, for example `LaptopApplication*`, and click **Search**. The disconnected application instance is displayed in the Search Results section.

3. Select **LaptopApplicationInstance** in the Search Results section, and then click **Select Children**. The Select Children page is displayed.

4. Select the required child attributes, as shown in Figure 1–2:

Figure 1–2 Child Attributes



Deployment Manager Wizard

Step 2 of 4

Select Children >>

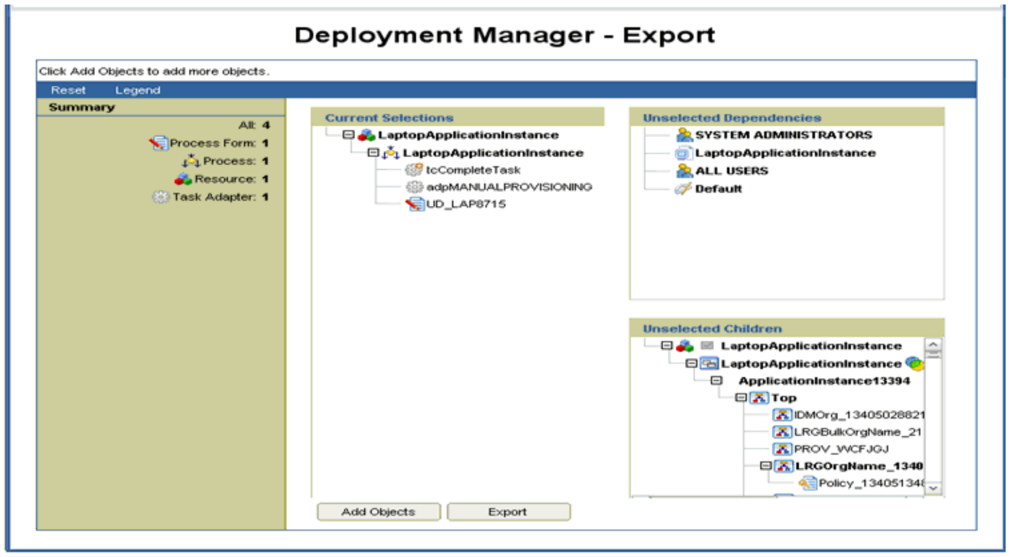
- ⊟ LaptopApplicationInstance
  - ⊟ ☑ LaptopApplicationInstance
    - ☑ tcCompleteTask
    - ☑ adpMANUALPROVISIONING
    - ☑ UD_LAP8715
  - ⊟ ☐ LaptopApplicationInstance
    - ⊟ ☐ ApplicationInstance13394
      - ⊟ ☐ Top
        - ☐ IDMOrg_1340502882155
        - ☐ LRGBulkOrgName_21340510897661
        - ☐ PROV_WCFJGJ

5. Click **Select Dependencies**. The Select Dependencies page is displayed.

6. Click **Confirmation**. In the Confirmation page, click **Add For Export**.

7. After verifying that all the required dependencies are displayed in the export summary, as shown in Figure 1–3, click **Export**.

Figure 1–3 Export Summary



Deployment Manager - Export

Click Add Objects to add more objects.

Summary

All: 4
Process Form: 1
Process: 1
Resource: 1
Task Adapter: 1

Current Selections

- ⊟ LaptopApplicationInstance
  - ⊟ LaptopApplicationInstance
    - tcCompleteTask
    - adpMANUALPROVISIONING
    - UD_LAP8715

Unselected Dependencies

- SYSTEM ADMINISTRATORS
- LaptopApplicationInstance
- ALL USERS
- Default

Unselected Children

- ⊟ LaptopApplicationInstance
  - ⊟ LaptopApplicationInstance
    - ⊟ ApplicationInstance13394
      - ⊟ Top
        - IDMOrg_13405028821
        - LRGBulkOrgName_21
        - PROV_WCFJGJ
        - ⊟ LRGOrgName_1340
          - Policy_13405134

8. Provide a name to the XML file, such as DisconnectedLaptopExp.xml. Upon successful export, a message is displayed.

1.2.3 Importing the Disconnected Application Instance in Production Environment

To import the disconnected application instance in production environment:

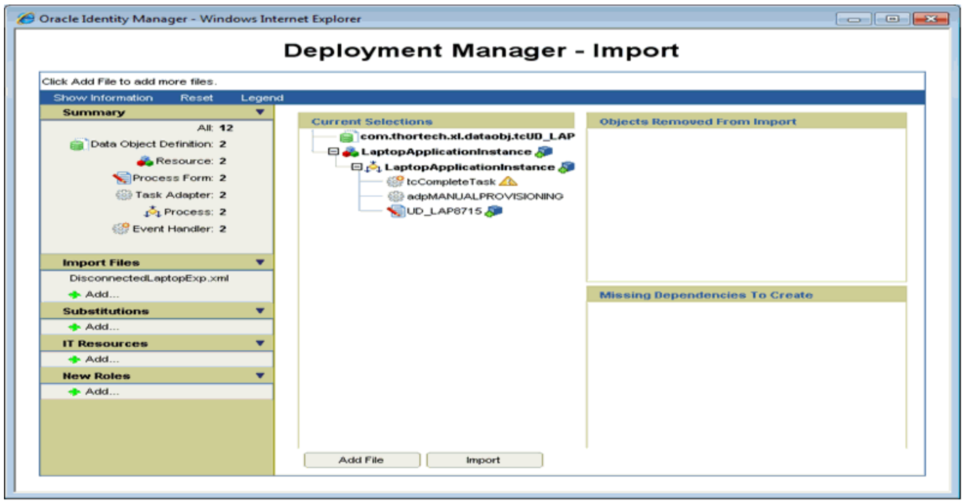
1. In the left pane of the Oracle Identity System Administration, under System Management, click **Import**.

2. Provide the path to the exported XML file, and then click **OK**. A confirmation page is displayed. Click **Add File**.

3. In the Substitutions page, you can provide substitutions for users or groups. If there are no substitutions, then click **Cancel Substitution**.

4. In the import summary, as shown in , check for any unresolved dependency, as shown in Figure 1–4 and then click **Import**.

Figure 1–4 Import Summary



Oracle Identity Manager - Windows Internet Explorer

Deployment Manager - Import

Click Add File to add more files.

Show Information Reset Legend

Summary

All: 12
Data Object Definition: 2
Resource: 2
Process Form: 2
Task Adapter: 2
Process: 2
Event Handler: 2

Import Files
DisconnectedLaptopExp.xml
Add...

Substitutions
Add...

IT Resources
Add...

New Roles
Add...

Current Selections
com.thortech.xl.dataobj.tcUD_LAP
LaptopApplicationInstance
LaptopApplicationInstance
tcCompleteTask
adpMANUALPROVISIONING
UD_LAP8715

Objects Removed From Import

Missing Dependencies To Create

Add File Import

5. Verify that the process definition, resource object, and forms have been successfully imported.

1.2.4 Modifying the Application Instance from Disconnected to Connected

In the environment where the application instance has been imported, make the following changes to convert the disconnected application instance to a connected application instance:

1. Login to the Design Console.

2. Expand **Resource Management**. Click **Resource Objects** to open the Resource Objects form.

3. Change the type of the resource object from Disconnected to **Application**.

4. Define new IT resource parameters in conjunction with the connected resource as required in the IT Resource Type Definition form.

5. Modify the existing IT resource (assuming that the ITResource is the same) with the new parameters added in step 4.

6. Expand Process Management, and click **Process Definition** to open the Process Definition form.

7. Search the process definition of the disconnected application instance. The following tasks are displayed:

   - ManualProvisioningStart
   - ManualProvisioningEnd
   - ManualEnableStart

- ManualEnableEnd
- ManualDisableStart
- ManualDisableEnd
- ManualRevokeStart
- ManualRevokeEnd

8. For each task, perform the following:
   a. Double-click the Task row to open the task details. See "Modifying Process Tasks" on page 2-10 for more information about modifying process tasks.
   b. Rename the task. For example, change the task name from ManualProvisioningStart to XXManualProvisioningStart.
   c. Make sure the **Conditional** option is selected. In addition, ensure that the **Required for Completion** option is not selected.
   d. If the task is an enable/disable/revoke task, then change the task effect to **No effect**.
   e. In the Integration tab, disassociate the adapters attached to the task by clicking on **Remove**.
   f. Remove task dependency, if any.
   g. Remove undo/recovery/generated tasks, if any.
   h. Change the object status mapping, if any, to none.

Note: Step 6a through 6g are to ensure that the existing tasks for disconnected application instance do not start when the application instance is exported as a connected application instance.

9. There is a task by the name *PARENT_FORM_NAME* Updated. This task triggers whenever the parent form is updated. Make sure to disassociate the existing adapters attached to the task and customize the task as required.
10. If there are any tasks related to the child form, then make sure to remove the triggers for create/update/delete by clicking **Clear**. If these tasks are not going to be reused, then disassociate the adapters attached to these tasks and rename the tasks to ensure that they do not run. Oracle recommends creating new tasks for each create, update, and delete trigger.

Note:

- Optionally, the same tasks for the child data can be retained but custom adapters must be defined for the create/update/delete trigger.
- For a disconnected application instance with child data, the task with the delete trigger will be associated with the tcCompleteTask adapter. Make sure to define and attach a custom adapter to this task to enable proper deletion of entitlement or child data.

11. Define custom adapters for the create, disable, enable, revoke, and update account tasks. If there are child tables, then make sure to define custom adapters for the same.

12. Create the following tasks in the process definition, and associate the corresponding adapters to each of those tasks. Map the required undo/recovery tasks and set the object status mapping.
    - **Create User:** Ensure that in the task properties, the **Required for Completion** option is selected and the **Conditional** option is not selected.
    - **Disable User:** Ensure that the task effect is Disable Processes or Access to Application.
    - **Enable User:** Ensure that the task effect is Enable Processes or Access to Application.
    - **Delete User:** Ensure that the task effect is Revoke Processes or Access to Application.
    - *ATTRIBUTE_NAME* **Updated:** For each attribute defined in the process form, corresponding update tasks have to be created. These tasks are triggered on updates to the process form, for example, Account Name Update, Account ID Updated, and so on.
13. If there is a child table, then define tasks for each trigger type, such as create, update, and delete.

Test the connected application instance by provisioning it to a few users in the test environment. You must define a new application instance with the modified resource object and IT resource to provision the application instance to users.

1.2.5 Testing the Connected Application Instance

After converting the disconnected application instance to a connected application instance:

- Export the modified resource from the test environment.
- Import the modified resource to the production environment.

2 Developing Provisioning Processes

This chapter describes process management with the Design Console. It contains the following topics:

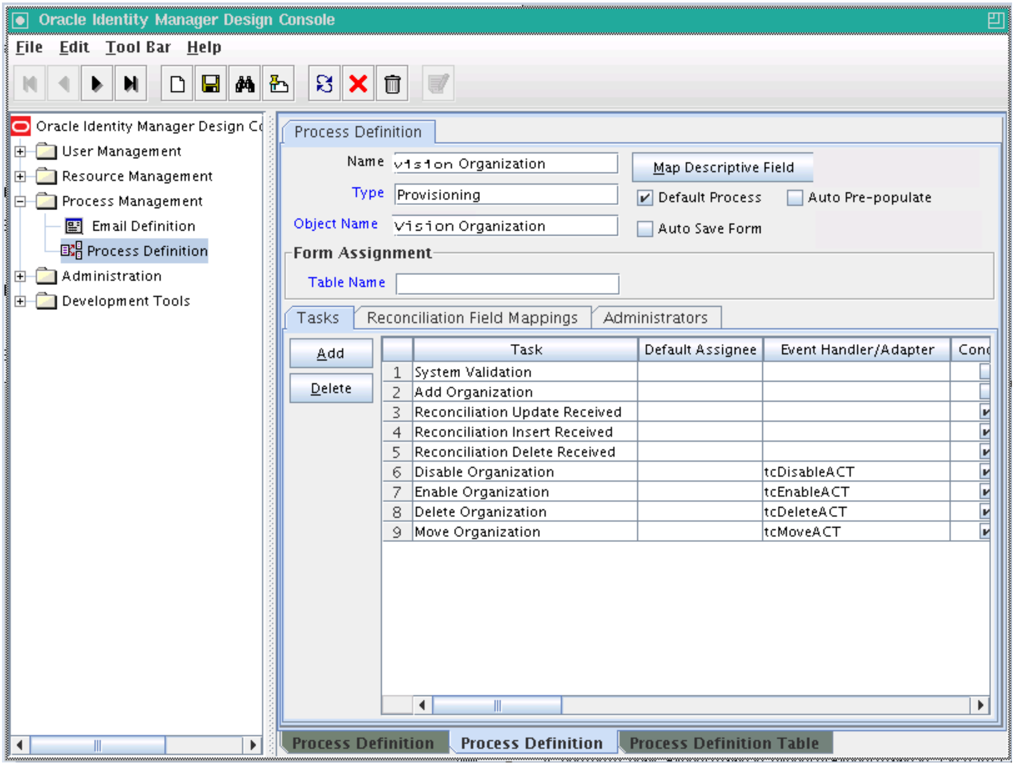
- Process Definition Form

2.1 Process Definition Form

A process is the mechanism for representing a logical workflow for provisioning in Oracle Identity Manager. Process definitions consist of tasks. Process tasks represent the steps that you must complete to fulfill the purpose of a process. For example, in a provisioning process, tasks are used to enable a user or organization to access the target resource.

The Process Definition form shown in Figure 2–1 is in the Process Management folder. You use this form to create and manage the provisioning processes that you associate with your resource objects.

Figure 2–1 Process Definition Form



Name: Vision Organization

Type: Provisioning

Object Name: Vision Organization

[✓] Default Process [ ] Auto Pre-populate

[ ] Auto Save Form

Table Name:

	Task	Default Assignee	Event Handler/Adapter	Cond
1	System Validation			
2	Add Organization			
3	Reconciliation Update Received			✓
4	Reconciliation Insert Received			✓
5	Reconciliation Delete Received			✓
6	Disable Organization		tcDisableACT	✓
7	Enable Organization		tcEnableACT	✓
8	Delete Organization		tcDeleteACT	✓
9	Move Organization		tcMoveACT	✓

In Figure 2–1, the **Xellerate Organization** provisioning process is created and assigned to the resource object of the same name.

Note: Not all the form columns are captured in Figure 2–1; additional field columns extend on the right of the Tasks table.

Table 2–1 describes the fields of the Process Definition form.

Table 2–1 Fields of the Process Definition Form

Field Name	Description
Name	The name of the process.
Type	The classification type of the process definition.
Object Name	The name of the resource object to which the process will be assigned.
Map Descriptive Field	Click this button to select a field that will be used as an identifier of the process definition after an instance is assigned to a resource object.
Render Workflow	Click this button to start a Web browser and display the current workflow definition by using the Workflow Renderer tool.
Default Process	This check box determines if the current process is the default provisioning process for the resource object with which it is associated. Select the check box to set the process as the default provisioning process for the resource object to which it is assigned. If you deselect the check box, the process will not be the default. It will only be invoked if a process selection rule causes it to be chosen.

Table 2–1 (Cont.) Fields of the Process Definition Form

Field Name	Description
Auto Pre-Populate	This check box designates whether the fields of a custom form are populated by Oracle Identity Manager or a user. Two types of forms are affected: ▪ Forms that are associated with the process ▪ Forms that contain fields with prepopulated adapters attached to them If the **Auto Pre-Populate** check box is selected, when the associated custom form is displayed, the fields that have prepopulate adapters attached to them will be populated by Oracle Identity Manager. When this check box is deselected, a user must populate these fields by clicking the **Pre-Populate** button on the toolbar or by manually entering the data. **Note**: This setting does not control the triggering of the prepopulate adapter. It only determines if the contents resulting from the execution of the adapter are displayed in the associated form field(s) because of Oracle Identity Manager or a user. For more information about prepopulate adapters, see "Working with Prepopulate Adapters" on page 3-25. **Note**: This check box is only relevant if you have created a process form that is to be associated with the process and prepopulate adapters are used with that form.
Table Name	The name of the table that represents the form that is associated with the process definition.

2.1.1 Creating a Process Definition

To create a process definition:

1. Open the Process Definition form.
2. In the **Name** field, type the name of the process definition.
3. Double-click the **Type** lookup field.

   From the Lookup dialog box that is displayed, select the classification type (Approval) of the process definition.
4. Double-click the **Object Name** lookup field.

   From the Lookup dialog box that is displayed, select the resource object that will be associated with the process definition.
5. Optional. Select the **Default Process** check box to make this the default provisioning process for the resource object to which it is assigned.

   If you do not want the current process definition to be the default, go to Step 6.
6. Optional. Select the **Auto Save Form** check box to suppress the display of the provisioning process' custom form and automatically save the data in it.

   This setting is only applicable to provisioning processes.

   To display provisioning process' custom form and solicit users for information, deselect this check box.

Note: If you select the **Auto Save Form** check box, ensure that all fields of the associated "custom" process form have adapters associated with them. However, a process form can have default data or object to the process data flow mapping or organization defaults.

For more information about adapters and their relationship with fields of custom forms, see Chapter 3, "Using the Adapter Factory".

7. If a custom form is to be associated with the process definition, this form contains fields that have prepopulate adapters attached to them, and you want these fields to be populated automatically by Oracle Identity Manager, select the **Auto Pre-Populate** check box.

   If the fields of this form are to be populated manually (by an user clicking the **Pre-Populate** button on the Toolbar), deselect the **Auto Pre-Populate** check box.

   Note: If the process definition has no custom form associated with it, or this form's fields have no pre-populate adapters attached to them, deselect the **Auto Pre-Populate** check box. For more information about prepopulate adapters, see "Working with Prepopulate Adapters" on page 3-25.

8. Double-click the **Table Name** lookup field.

   From the Lookup window that is displayed, select the table that represents the form associated with the process definition.

9. Click **Save**.

   The process definition is created and the **Map Descriptive Field** button is enabled. If you click this button, the Map Descriptive Field dialog box is displayed.

   From this window, you can select the field (for example, the Organization Name field) that will be used as an identifier of the process definition when an instance of the process is assigned to a resource object. This field and its value will be displayed in the reconciliation Manger form.

   Note: If a process has a custom process form attached to it, the fields on that form will also be displayed in this window and be available for selection.

2.1.2 Tabs on the Process Definition Form

After you start the Process Definition form and create a process definition, the tabs of this form become functional.

The Process Definition form contains the following tabs:

- Tasks Tab
- Reconciliation Field Mappings Tab

Each of these tabs is described in the following sections.

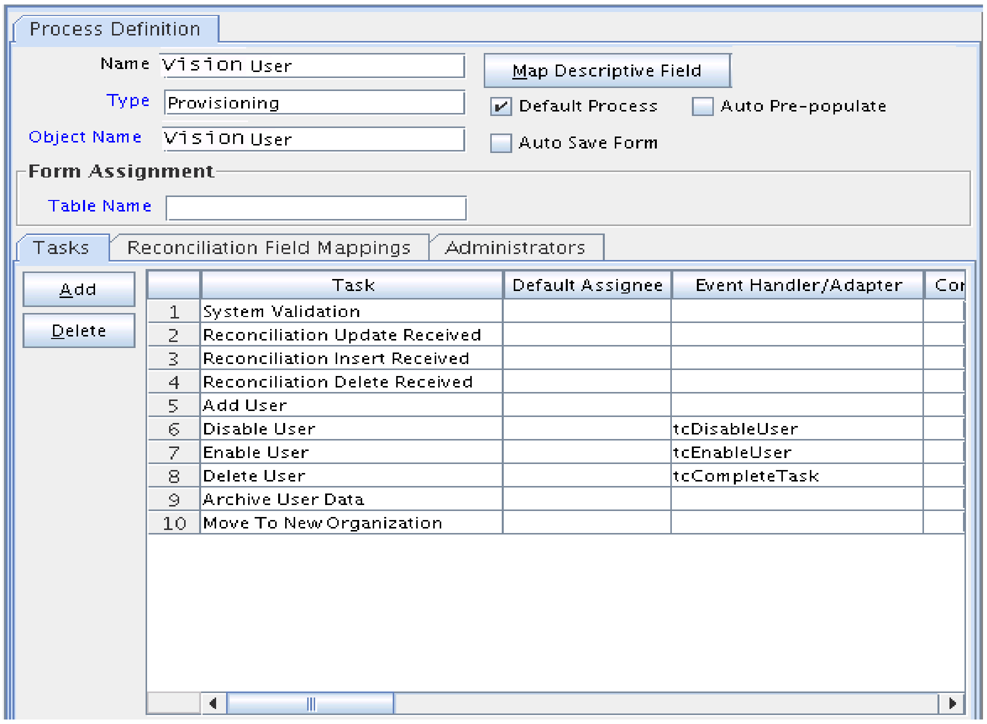
2.1.2.1 Tasks Tab

You use this tab to:

- Create and modify the process tasks that comprise the current process definition
- Remove a process task from the process definition (when it is no longer valid)

Figure 2–2 displays the Tasks tab of the Process Definition form.

Figure 2–2 Tasks Tab of the Process Definition Form



Process Definition

Name: Vision User
Type: Provisioning
Object Name: Vision User

Map Descriptive Field

[✓] Default Process
[ ] Auto Pre-populate
[ ] Auto Save Form

Form Assignment

Table Name:

Tasks | Reconciliation Field Mappings | Administrators

Add
Delete

	Task	Default Assignee	Event Handler/Adapter	Co
1	System Validation			
2	Reconciliation Update Received			
3	Reconciliation Insert Received			
4	Reconciliation Delete Received			
5	Add User			
6	Disable User		tcDisableUser	
7	Enable User		tcEnableUser	
8	Delete User		tcCompleteTask	
9	Archive User Data			
10	Move To New Organization			

See Also: See "Modifying Process Tasks" on page 2-10 for information about editing process tasks

---

2.1.2.1.1 Adding a Process Task

Process tasks represent the steps that you must complete in a process.

To add a process task:

1. Click **Add**.

   The Creating New Task dialog box is displayed.

2. In the **Task Name** field, enter the name of the process task.
3. From the Toolbar of the Creating New Task window, click **Save**. Then, click **Close**.

   The process task is added to the process definition.

2.1.2.1.2 Editing a Process Task

For instructions about how to edit and set process tasks, see "Modifying Process Tasks" on page 2-10.

2.1.2.1.3 Deleting a Process Task

To delete a process task:

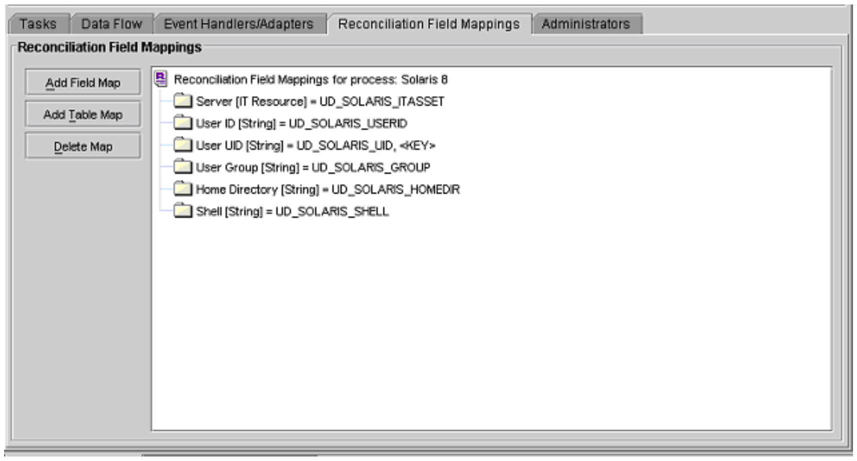
1. Select the process task that you want to delete.
2. Click **Delete**.

   The process task is removed from the process definition.

2.1.2.2 Reconciliation Field Mappings Tab

You use the Reconciliation Field Mappings tab shown in Figure 2–3 to define a relationship between data elements in a target system or trusted source and fields in Oracle Identity Manager.

Figure 2–3 Reconciliation Field Mappings Tab of the Process Definition Form



Reconciliation Field Mappings for process: Solaris 8
- Server [IT Resource] = UD_SOLARIS_ITASSET
- User ID [String] = UD_SOLARIS_USERID
- User UID [String] = UD_SOLARIS_UID, <KEY>
- User Group [String] = UD_SOLARIS_GROUP
- Home Directory [String] = UD_SOLARIS_HOMEDIR
- Shell [String] = UD_SOLARIS_SHELL

Only fields that you define in the **Reconciliation Fields** tab of the associated resource are available for mapping. Using a reconciliation event, these mappings determine which fields in Oracle Identity Manager to populate with information from the target system. For target resources (not trusted sources), you can use this tab to indicate which fields are key fields. Key fields determine the values that must be same on the process form and the reconciliation event to generate a match on the **Processes Matched Tree** tab of the Reconciliation Manager form.

For each mapping, the following information is displayed:

- Name of the field, as defined on the **Reconciliation Fields** tab of the associated resource, on the target system or trusted source that is to be reconciled with data in Oracle Identity Manager.
- Data type associated with the field, as defined on the **Reconciliation Fields** tab of the associated resource.

  Possible values are **Multi-Valued**, **String**, **Number**, **Date**, and **IT resource**.

  Note: The IT Resource must be marked as a key field.

- **For trusted sources**: For user discovery, mapping of the data in the trusted source field to the name of a field on the users form, or for organization discovery, mapping of the data in the trusted source field to the name of a field on the Oracle Identity Manager Organizations form.

  If you are performing user and organization discovery with a trusted source, organization discovery must be conducted first.

- **For target resources**: The name of the field on the resource's custom (provisioning) process form to which the data in the target resources field is to be mapped.
- **For target resources**: Indicator designating if the field is a key field in the reconciliation for this target resource.

  For provisioning processes to match a reconciliation event data, the key field values in their process forms must be the same as those in the reconciliation event.

> **Note:** Oracle recommends configuring both the entitlement attribute and the key attribute for the child data in reconciliation field mappings to enable effective duplicate entitlement or child data validation. See "Duplicate Validation for Entitlements or Child Data" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager* for information about duplicate validation for entitlements or child data.

2.1.2.2.1 User Account Status Reconciliation

To configure user account status reconciliation, you must do the following:

- **For trusted sources**: You must create a reconciliation field, for example, `Status`, in the corresponding trusted resource object, which denotes the status of the user in the target. The value of this field must be either `Active` or `Disabled`. This reconciliation field must be mapped to the user attribute `status` in the corresponding process definition.
- **For target resources**: You must create a reconciliation field, for example, `Status`, in the corresponding resource object, which denotes the status of the resource in the target. This reconciliation field must be mapped to the process attribute `OIM_OBJECT_STATUS` in the corresponding process definition. The following statuses are supported for target resource reconciliation:
  - Revoked
  - Provisioned
  - Ready
  - Provide Information
  - Enabled
  - None
  - Waiting
  - Provisioning
  - Disabled

2.1.2.2.2 Mapping a Target Resource Field to Oracle Identity Manager

You can map the fields on a target resource or trusted source, as defined on the **Reconciliation Fields** tab of the associated resource definition, to applicable fields in Oracle Identity Manager. These mappings determine the fields that must be updated in Oracle Identity Manager in a reconciliation event. These mappings occur when you click one of the following on the Reconciliation Manager form:

- The **Create User** or **Create Organization** button
- The **Link** button on the **Matched Users** or **Matched Organizations** tab

- The **Establish Link** button on the **Processes Matched Tree** tab

For user discovery on a trusted source, you define the fields to be mapped from the **User** resource to fields in the User provisioning process. The fields (that is, the user attributes) to which you will map your trusted source fields are derived from the Users form.

For organization discovery on a trusted source, you define fields to be mapped from the Oracle Identity Manager Organization resource to fields in the Oracle Identity Manager Organization provisioning process. The fields (that is, the organization attributes) to which you will map your trusted source fields are derived from the Organizations form.

After you have accessed the provisioning process definition for the associated resource and selected the **Reconciliation Field Mappings** tab, use one of the two procedures described in the following sections.

Mapping a Single Value Field

To map a single value field:

1. Click **Add Field Map**.

   The Add Reconciliation Field Mappings dialog box is displayed.

2. Select the field on the target system that you want to map from the menu in the Field Name field.

   Oracle Identity Manager will automatically supply the field type based on what was entered for this field on the associated **Resource Object** form.

3. For trusted sources:

   Select a value from the **User Attribute** menu and click **OK**. Go to Step 4.

   For target resources:

   Double-click **Process Data Field**. Select the correct mapping from the **Lookup** dialog box and click **OK**.

4. If you are defining mapping for a trusted source, go to step 5.

   Set the **Key Field for Reconciliation Matching** check box for target resources only. If this check box is selected, Oracle Identity Manager evaluates if the value of this field on the provisioning process form matches the value of the field in the reconciliation event. All matched processes are displayed on the **Processes Matched Tree** tab of the Reconciliation Manager form. If this check box is deselected, Oracle Identity Manager does not require the value of this field to match the process form and reconciliation event for process matching.

   Note: To set a field as a key field, it must be set as required on the **Object Reconciliation** tab of the applicable resource.

5. Click **Save**.

   The mapping for the selected fields is applied the next time a reconciliation event is received from the target resource or trusted source.

Mapping a Multi-Value Field (For Target Resources Only)

To map a multi-value field:

1. Click **Add Table Map**.

The Add Reconciliation Table Mappings dialog box is displayed.

2. Select the multi-value field on the target system that you want to map from the menu in the **Field Name** field.

   Oracle Identity Manager will automatically supply the field type based on what was entered for this field on the associated Resource Object form.

3. Select the child table you defined on the target resource's process form from the **Table Name** menu.

4. Double-click **Process Data Field**, and select the correct mapping from the Lookup dialog box, and click **OK**.

5. Save and close the Add Reconciliation Table Mappings dialog box.

6. Right-click the multi-value field you just mapped, and select Define a property field map from the menu that is displayed.

7. Select the component (child) field you want to map.

   Oracle Identity Manager will automatically supply the field type based on what was entered for this field on the associated Resource Object form.

8. Double-click the **Process Data Field** field.

   Select the correct mapping from the Lookup dialog box and click **OK**.

9. Set the **Key Field for Reconciliation Matching** check box.

   If this check box is selected, Oracle Identity Manager compares the field value on the provisioning process child form with the field value in the reconciliation event. All matching processes are displayed on the **Processes Matched Tree** tab of the Reconciliation Manager form. If you deselect this check box, the value of this field does not have to match on the process form and reconciliation event for process matching. Ensure that at least one component (child) field of each multi-valued field is set as a key field. This improves the quality of the matches generated on the **Process Matched Tree** tab.

   Note: Key fields must be set as required on the **Object Reconciliation** tab of the applicable resource.

10. Repeat Steps 6 through 9 for each component (child) field defined on the multi-value field.

11. Click **Save**.

    The mapping for the selected fields will be applied the next time a reconciliation event is received from the target resource.

2.1.2.2.3 Deleting a Mapping

This procedure is used to delete a mapping that has been established between a field in Oracle Identity Manager and a field on the target system or trusted source as defined on the **Reconciliation Fields** tab of the associated resource definition.

To delete a mapping:

1. Go to the provisioning process definition for the associated resource.
2. Select the **Reconciliation Field Mappings** tab.
3. Select the field mapping you want to delete.

4. Click **Delete Map**.

   The mapping for the selected field is deleted.

2.1.3 Modifying Process Tasks

To modify a process task for a process definition, double-click its row heading. The Editing Task window is displayed, containing additional information about the process task.

The Editing Task window contains the following tabs:

- General Tab
- Integration Tab
- Task Dependency Tab
- Responses Tab
- Task to Object Status Mapping Tab

Note: You must not modify the Xellerate Users process definition.

2.1.3.1 General Tab

You use this tab to set high-level information for the task that you want to modify. For this example, the **Create User** task is used to create a user in the Solaris environment.

Table 2–2 describes the fields of the General tab.

Table 2–2 Fields of the General Tab

Field Name	Description
Task Name	The name of the process task.
Task Description	Explanatory information about the process task.
Duration	The expected completion time of the current process task in days, hours, and minutes.
Conditional	This check box determines if a condition is met to add the current process task to the process. Select this check box to prevent the process task from being added to the process unless a condition has been met. Clear this check box to not require the condition to be met for the process task to be added to the process.
Required for Completion	This check box determines if the current process task must be completed for the process to be completed. Select this check box to require the process task to have a status of Completed before the process can be completed. Deselect this check box to ensure that the status of the process task does not affect the completion status of the process.
Constant Duration	Not applicable
Task Effect	From this box, select the process action you want to associate with the task, for example, disable or enable. A process can enable or disable a user's access to a resource. When the disable action is chosen, all tasks associated with the disable action are inserted. **Note**: If you do not want the process task to be associated with a particular process action, select **No Effect** from the box.

Table 2–2 (Cont.) Fields of the General Tab

Field Name	**Description**
Disable Manual Insert	This check box determines if a user can manually add the current process task to the process. Select this check box to prevent the process task from being added to the process manually. Deselect this check box to enable a user to add the process task to the process.
Allow Cancellation while Pending	This check box determines if the process task can be canceled if its status is Pending. Select this check box to allow the process task to be canceled if it has a Pending status. Deselecting this check box to prevent the process task from being canceled if its status is Pending.
Allow Multiple Instances	This check box determines if the process task can be inserted into the current process more than once. Select this check box to enable multiple instances of the process task to be added to the process. Deselect this check box to enable the process task to be added to the current process only once.
Retry Period in Minutes	If a process task is rejected, this field determines the interval before Oracle Identity Manager inserts a new instance of that task with the status of Pending. When the value of the Retry Period in Minutes field is **30**, it means that if the Create User process task is rejected, then in 30 minutes Oracle Identity Manager adds a new instance of this task and assigns it a status of Pending. **Note:** If you specify a value for this field, then you must ensure the following: ■ The Task Timed Retry scheduled job is not disabled. See the "Predefined Scheduled Tasks" section in *Oracle Fusion Middleware Administrator's Guide* for more information. ■ Frequency of the Task Timed Retry scheduled job is less than or equal to value of this field. ■ The **Allow Multiple Instances** checkbox of the process task that is being retried must be selected.
Retry Count	Determines how many times Oracle Identity Manager retries a rejected task. When the value of the Retry Count field is **5**, it means that if the Create User process task is rejected, then Oracle Identity Manager adds a new instance of this task, and assigns it a status of Pending. When this process task is rejected for the fifth time, Oracle Identity Manager no longer inserts a new instance of it.

Table 2–2 (Cont.) Fields of the General Tab

Field Name	Description
Child Table/ Trigger Type	These boxes specify the action that Oracle Identity Manager performs in the child table of a custom form that is associated with the current process, as indicated by the **Table Name** field of the **Process Definition** form. From the **Child Table** box, select the child table of the custom form where Oracle Identity Manager will perform an action. From the Trigger Type box, specify the action that Oracle Identity Manager is to perform in the child table. These actions include: ■ **Insert**. Adds a new value to the designated column of the child table ■ **Update**. Modifies an existing value from the corresponding column of the child table ■ **Delete**. Removes a value from the designated column of the child table **Note**: If the custom process form does not have any child tables associated with it, the **Child Table** box will be empty. In addition, the Trigger Type box will be grayed out.
Off-line	This flag is applicable only for user attribute propagation tasks. If the flag is set for a user attribute propagation task, the task insertion is asynchronous.

2.1.3.1.1 Modifying a Process Task's General Information

To modify the general information for a process task:

1. Double-click the row heading of the task you want to modify.

   The Editing Task dialog box is displayed.

2. Click the **General** tab.
3. In the **Description** field, enter explanatory information about the process task.
4. Optional. In the **Duration** area, enter the expected completion time of the process task (in days, hours, and minutes).
5. If you want a condition to be met for the process task to be added to the Process Instance, select the **Conditional** check box. Otherwise, go to Step 6.

   Note: If you select the **Conditional** check box, you must specify the condition to be met for the task to be added to the process.

6. When you want the completion status of the process to depend on the completion status of the process task, select the **Required for Completion** check box.

   By doing so, the process cannot be completed if the process task does not have a status of Completed.

   If you do not want the status of the process task to affect the completion status of the process, go to Step 7.

7. To prevent a user from manually adding the process task into a currently running instance of the process, select the **Disable Manual Insert** check box. Otherwise, go to Step 8.

8. To enable a user to cancel the process task if its status is Pending, select the **Allow Cancellation while Pending** check box. Otherwise, go to Step 9.
9. To allow this task to be inserted multiple times in a single process instance, select the **Allow Multiple Instances** check box. Otherwise, go to Step 10.
10. Click the **Task Effect** box.

    From the custom menu that is displayed, select one of the following:

    - **Enable Process or Access to Application**. If a resource is reactivated by using the enable function, all tasks with this effect are inserted into the process. If you select this option, you must also select the **Allow Multiple Instances** check box.
    - **Disable Process or Access to Application**. If a resource is deactivated by using the disable function, all tasks with this effect are inserted into the process. If you select this option, you must also select the **Allow Multiple Instances** check box.
    - **Revoke Process or Access to Application**. When the resource is revoked, the revoke workflow is executed without canceling the existing tasks in the provisioning process.
    - **No Effect**. This is the default process action associated with all tasks. If this option is selected, the task is only inserted during normal provisioning unless it is conditional.
11. Optional. If the process task is **Rejected**, you might want Oracle Identity Manager to insert a new instance of this process task (with a status of **Pending**).

    For this to occur, enter a value in the **Retry Period in Minutes** field. This designates the time in minutes that Oracle Identity Manager waits before adding this process task instance.

    In the **Retry Count** field, enter the number of times Oracle Identity Manager will retry a rejected task. For example, suppose **3** is displayed in the **Retry Count** field. If the task is rejected, Oracle Identity Manager adds a new instance of this task, and assigns it a status of Pending. After this process task is rejected for the fourth time, Oracle Identity Manager no longer inserts a new instance of the process task.

    Note: If either **Retry Period** or **Retry Count** is selected, you must specify parameters for the other option because they are both related.

12. From the **Child Table** box, select the child table of the custom form where Oracle Identity Manager will perform an action.

    From the **Trigger Type** box, specify the action that Oracle Identity Manager will perform in the child table. These actions include the following:

    - **Insert**: Adds a new value to the designated column of the child table
    - **Update**: Modifies an existing value from the corresponding column of the child table
    - **Delete**: Removes a value from the designated column of the child table

    Note: If the custom process form does not have any child tables associated with it, the **Child Table** box will be empty. In addition, the **Trigger Type** box will be grayed out.

13. Click **Save**.

    The modifications to the process task's top-level information reflects the changes you made in the **General** tab.

2.1.3.1.2 Triggering Process Tasks for Events Defined in Lookup.USR_PROCESS_TRIGGERS Fields

When a user attribute is defined in Lookup.USR_PROCESS_TRIGGERS, for each modification of the attribute, the corresponding process task is triggered for each provisioned resource. This is same for the First Name, Last Name, Display Name (USR_DISPLAY_NAME) user attributes and custom user attributes. However, for the Lookup.USR_PROCESS_TRIGGERS fields USR_STATUS, USR_LOCKED, USR_LOCKED_ON, and USR_MANUALLY_LOCKED, the attached process task is not triggered.

The following sections describe how to trigger the process tasks for the Lookup.USR_PROCESS_TRIGGERS fields:

For the USR_STATUS Attribute

It is not possible to run a task via Lookup.USR_PROCESS_TRIGGERS for the USR_STATUS attribute because this attribute is processed separately by Oracle Identity Manager. This attribute is changed by enabling, disabling, or deleting a user. These operations have a special effect on the provisioned resources because the corresponding process tasks are started via the Task Effect setting, as described in Table 2–2, " Fields of the General Tab". For these three operations, the Lookup.USR_PROCESS_TRIGGERS is not used. Therefore, when the status changes, perform the following to run the process task:

For transition from Disabled to Enabled status:

1. In the Process Definition form, create a process task named `Enable User`.
2. Open the Editing Task window, and click the **General** tab.
3. From the Task Effect list, select **Enables Process or Access to Application**.
4. Select **Conditional** and specify the condition to be met for the task to be added to the process.

For transition from Enabled to Disabled status:

1. In the Process Definition form, create a process task named `Disable User`.
2. Open the Editing Task window, and click the **General** tab.
3. From the Task Effect list, select **Enables Process or Access to Application**.
4. Select **Conditional** and specify the condition to be met for the task to be added to the process.

For transition from Enabled/Disabled/Provisioned to Revoked status:

1. In the Process Definition form, create a process task named `Delete User`.
2. Then set this task as an Undo task for the Create User task, which is the task that creates the user and is typically unconditional.
3. Select **Conditional** and specify the condition to be met for the task to be added to the process.

Note: when the Oracle Identity Manager user is deleted, for each completed task in each resource, Oracle Identity Manager tries to run the Undo tasks.

For the USR_LOCKED, USR_LOCKED_ON, USR_MANUALLY_LOCKED Attributes

The lock and unlock operations, are handled in Oracle Identity Manager as separate orchestrations. The orchestration is on:

```
entity-type="User" operation="LOCK"
```

Or:

```
entity-type="User" operation="UNLOCK"
```

The event handler that does the evaluation for Lookup.USR_PROCESS_TRIGGERS is:

```
oracle.iam.transUI.impl.handlers.TriggerUserProcesses
```

This is triggered only in the following user orchestrations:

- **MODIFY:** For generic fields
- **CHANGE_PASSWORD**, **RESET_PASSWORD:** For USR_PASSWORD propagation
- **ENABLE**, **DISABLE**, **DELETE**: For handling the execution of process tasks

For lock/unlock operations, the TriggerUserProcesses event handler is not triggered. Therefore, for the attributes modified through lock/unlock operations, the Lookup.USR_PROCESS_TRIGGERS is not checked.

If you want to run custom code for these operations when these fields are changed, then you can create event handlers and register them on the orchestrations mentioned in this section.

2.1.3.2 Integration Tab

By using the **Integration** tab, you can:

- Automate a process task by attaching an event handler or task adapter to it.
- Map the variables of the task adapter, so Oracle Identity Manager can pass the appropriate information when the adapter is triggered. This occurs when the process task's status is Pending.
- Break the link between the adapter handler and the process task, once the adapter or event handler is no longer applicable with the process task.

For example, suppose that the adpSOLARISCREATEUSER adapter is attached to the Create User process task. This adapter has nine adapter variables, all of which are mapped correctly as indicated by the `Y` that precedes each variable name.

Note:

- Event handlers are preceded with tc (Thor class), such as `tcCheckAppInstalled`. These are event handlers that Oracle provides. Customer-created event handlers cannot have a tc prefix in their name. Adapters are preceded with adp, for example, `adpSOLARISCREATEUSER`.
- From the Design Console, you cannot create or modify DOB event handlers. You can only view the existing event handlers.

See Also: Chapter 3, "Using the Adapter Factory" and Chapter 18, "Developing Event Handlers" for more information about adapters and event handlers

2.1.3.2.1 Assigning an Adapter or Event Handler to a Process Task

The following procedure describes how to assign an adapter or event handler to a process task.

Important: If you assign an adapter to the process task, the adapter will not work until you map the adapter variables correctly. See "Mapping Adapter Variables" on page 2-17 for details.

To assign an adapter or event handler to a process task:

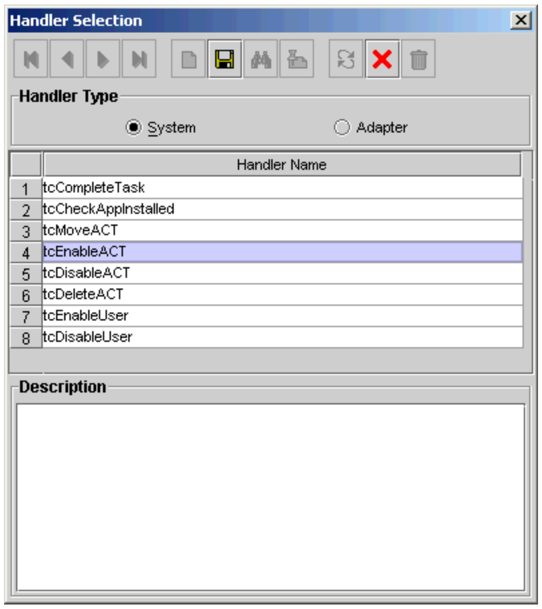
1. Double-click the row heading of the process task to which you want to assign an event handler or adapter.

   The Editing Task window is displayed.
2. Click the **Integration** tab.
3. Click **Add**.

   The **Handler Selection** dialog box is displayed, as shown in Figure 2–4.
4. To assign an event handler to the process task, select the **System** option.

   To add an adapter to the process task, select the **Adapter** option. A list of event handlers or adapters, which you can assign to the process task, is displayed in the **Handler Name** region.

Figure 2–4 Handler Selection Dialog Box



	Handler Name
1	tcCompleteTask
2	tcCheckAppInstalled
3	tcMoveACT
4	tcEnableACT
5	tcDisableACT
6	tcDeleteACT
7	tcEnableUser
8	tcDisableUser

5. Select the event handler or adapter that you want to assign to the process task.
6. From the Handler Selection window's Toolbar, click **Save**.

   A confirmation dialog box is displayed.
7. Click **OK**.

   The event handler or adapter is assigned to the process task.

2.1.3.2.2 Mapping Adapter Variables

> **See Also:** "Adapter Mapping Information" on page 3-34 for more information about the items to select in this procedure

---

> **Note:** To trigger a task associated with a change to a parent form field, the name of the task must be *field* Updated, where *field* is the name of the parent form field. If the task is not named according to this convention, it is not triggered during a field update.

To map an adapter variable:

1. Select the adapter variable that you want to map.
2. Click **Map**.

   The Data Mapping for Variable window is displayed.
3. Complete the **Map To**, **Qualifier, IT Asset Type**, **IT Asset Property**, **Literal Value**, and **Old Value** fields.

Note: IT Asset Type and IT Asset Property are displayed only when It Resources is selected from the Map To operations. The Literal Value field is displayed only when Literal is selected from Map To. Old Value check box is enabled only when Organization Definition or User Definition is selected from Map To.

4. From the Data Mapping for Variable window's Toolbar, click **Save**.
5. Click **Close**.

   The mapping status for the adapter variable changes from `N` to `Y`. This indicates that the adapter variable has been mapped.

2.1.3.2.3 Removing an Adapter or Event Handler from a Process Task

To remove an adapter or event handler from a process task:

1. Click **Remove**.

   A confirmation dialog box is displayed.
2. Click **OK**.

   The event handler or adapter is removed from the process task.

2.1.3.3 Task Dependency Tab

You use the **Task Dependency** tab to determine the logical flow of process tasks in a process. Through this tab, you can:

- Assign **preceding** tasks to a process task.

  These tasks must have a status of Completed before Oracle Identity Manager or a user can trigger the current process task.
- Assign **dependent** tasks to a process task.

  Oracle Identity Manager or a user can trigger these tasks only after the current process task has a status of Completed.
- Break the link between a preceding task and the current task so that the preceding task's completion status no longer has any effect on the current task being triggered.
- Break the link between the current task and a dependent task so that the current task's completion status no longer has any bearing on triggering the dependent tasks.

For example, the **Create User** process task does not have any preceding tasks. Oracle Identity Manager triggers this task whenever the task is inserted into a process (for example, when an associated resource is requested). The **Create User** process task has seven dependent tasks. Before completion of this process task, each dependent task will have a status of **Waiting**. Once this task achieves a status of Completed, each of these process tasks are assigned a status of **Pending**, and Oracle Identity Manager can trigger them.

2.1.3.3.1 Assigning a Preceding Task to a Process Task

To assign a preceding task to a process task:

1. Double-click the row heading of the process task to which you want to assign a preceding task.

The **Editing Task** window is displayed.

2. Click the **Task Dependency** tab.
3. From the Preceding Tasks region, click **Assign**.

   The **Assignment** window is displayed.
4. From this window, select the preceding task, and assign it to the process task.
5. Click **OK**.

   The preceding task is assigned to the process task.

2.1.3.3.2 Removing a Preceding Task from a Process Task

To remove a preceding task from a process task:

1. Select the preceding task that you want to delete.
2. From the Preceding Tasks region, click **Delete**.

   The preceding task is removed from the process task.

2.1.3.3.3 Assigning a Dependent Task to a Process Task

To assign a dependent task to a process task:

1. Double-click the row heading of the process task to which you want to assign a dependent task.

   The Editing Task window is displayed.
2. Click the **Task Dependency** tab.
3. From the **Dependent Tasks** region, click **Assign**.

   The Assignment window is displayed.
4. From this window, select the dependent task, and assign it to the process task.
5. Click **OK**.

   The dependent task is assigned to the process task.

2.1.3.3.4 Removing a Dependent Task from a Process Task

To remove a dependent task from a process task:

1. Select the dependent task that you want to delete.
2. From the **Dependent Tasks** region, click **Delete**.

   The dependent task is removed from the process task.

2.1.3.4 Responses Tab

You use the Responses tab to do the following:

- Define the response codes that can be received in conjunction with the execution of a particular process tasks. You can use response codes to represent specific conditions on the target system.
- Define the conditional tasks that are started if a response code is received during execution of this process task. These tasks are called generated tasks.
- Remove a response from a process task.
- Remove a generated task from a process task.

For example, when a Create User process task is completed, the `SUCCESS` response is activated. This response displays a dialog box with the message "The user was created successfully." In addition, Oracle Identity Manager triggers the Enable User process task.

Note: By default, the UNKNOWN response is defined for each process task that is rejected. This way, even when the system administrator does not add any responses to a process task, if this task is rejected, the user will be notified in the form of an error message in a dialog box.

2.1.3.4.1 Adding a Response to a Process Task

To add a response to a process task:

1. Double-click the row heading of the process task to which you want to add a response.

   The Editing Task window is displayed.
2. Click the **Responses** tab.
3. In the **Responses** region, click **Add**.

   A blank row is displayed in the Responses region.
4. Enter information in the **Response** field.

   This field contains the response code value. This field is case-sensitive.
5. Enter information in the **Description** field. This field contains explanatory information about the response.

   If the process task triggers the response, this information is displayed in the task information dialog box.
6. Double-click the **Status** lookup field.

   From the Lookup window that is displayed, select a task status level. If the response code is received, it will cause the task to be set to this status.
7. Click **Save**.

   The response you added would now reflect the settings you have entered.

2.1.3.4.2 Removing a Response from a Process Task

To remove a response from a process task:

1. Select the response that you want to delete.
2. From the **Responses** region, click **Delete**.

   The response is removed from the process task.

Note: You will not be able to delete a response from a process task that is invoked for any provisioning instance, even if the response is existing or is newly added. However, if the process task is not invoked for any provisioning instance, you will be able to delete the response.

2.1.3.4.3 Assigning a Generated Task to a Process Task

To assign a generated task to a process task:

1. Double-click the row heading of the process task to which you want to assign a generated task.

   The Editing Task window is displayed.
2. Click the **Responses** tab.
3. Select the response code for which you want to assign generated tasks.
4. From the **Tasks to Generate** region, click **Assign**.

   The Assignment window is displayed.
5. From this window, select the generated task, and assign it to the process task response.
6. Click **OK**.

   The generated task is assigned to the process task.

2.1.3.4.4 Removing a Generated Task From a Process Task

To remove a generated task from a process task:

1. Select a response code.
2. Select the generated task that you want to delete.
3. From the Tasks to Generate region, click **Delete**.

   The generated task is removed from the process task.

2.1.3.5 Task to Object Status Mapping Tab

A resource object contains data that is used to provision resources to users and applications.

In addition, a resource object is provided with predefined provisioning statuses, which represent the various statuses of the resource object throughout its life cycle as it is being provisioned to the target user or organization.

Note: Provisioning statuses are defined in the **Status Definition** tab of the **Resource Objects** form.

The provisioning status of a resource object is determined by the status of its associated provisioning processes, and the tasks that comprise these processes. For this reason, you must provide a link between the status of a process task and the provisioning status of the resource object to which it is assigned.

The **Task to Object Status Mapping** tab is used to create this link. Also, when this connection is no longer required, or you want to associate a process task status with a different provisioning status for the resource object, you must break the link that currently exists.

For this example, there are five mappings among process task statuses and provisioning statuses of a resource object. When the Create User process task achieves a status of `Completed`, the associated resource object will be assigned a provisioning status of `Provisioned`. However, if this task is canceled, the provisioning status for the resource object will be `Revoked`. `None` indicates that this status has no effect on the provisioning status of the resource object.

The following sections describe how to map a process task status to a provisioning status and unmap a process task status from a provisioning status.

2.1.3.5.1 Mapping a Process Task Status to a Provisioning Status

To map an process task status to a provisioning status:

1. Double-click the row heading of the process task, which has a status that you want to map to the provisioning status of a resource object.

   The Editing Task window is displayed.
2. Click the **Task to Object Status Mapping** tab.
3. Select the desired process task status.
4. Double-click the **Object Status** lookup field.

   From the Lookup window that is displayed, select the provisioning status of the resource object to which you want to map the process task status.
5. Click **OK**.

   The provisioning status you selected is displayed in the Task to Object Status Mapping tab.
6. Click **Save**.

   The process task status is mapped to the provisioning status.

2.1.3.5.2 Unmapping a Process Task Status From a Provisioning Status

To unmap an process task status from a provisioning status:

1. Select the desired process task status.
2. Double-click the **Object Status** lookup field.

   From the Lookup window that is displayed, select None. None indicates that this status has no effect on the provisioning status of the resource object.
3. Click **OK**.

   The provisioning status of None is displayed in the **Task to Object Status Mapping** tab.
4. Click **Save**.

   The process task status is no longer mapped to the provisioning status of the resource object.

Part II

Connectors

This part familiarizes you with tools and features for Oracle Identity Manager developers, and provides some simple examples to illustrate the concepts.

It contains the following chapters:

3 Using the Adapter Factory

Adapters are Java programs that enable you to integrate Oracle Identity Manager with other software solutions. This chapter describes how to create adapters using the Adapter Factory form. It contains these sections:

3.1 Introduction to Adapters

To be effective, it must be possible to integrate an access rights management application, such as Oracle Identity Manager, with other software solutions. This is necessary not only because there are many resources, but also because there is no single integration standard for connecting to these resources.

The traditional way to tackle this challenge is by using the common functionality that is supported by all the integrations. To do this, you need developers who can write this code. In addition, every time an existing software resource is modified, or a new one is added, you must write more code.

The Adapter Factory is a code-generation tool provided by Oracle Identity Manager. It helps you create Java classes, known as adapters, that simplify the integration challenge.

Note: Oracle Identity Manager can connect to external systems such as databases and directory servers by using Java APIs for JDBC and LDAP. In addition, for all other APIs, such as C, C++, VB, and COM/DCOM, you can create a Java wrapper so that Oracle Identity Manager can communicate with the API directly.

A resource has an associated provisioning process, which in turn has various tasks associated with it. Each task in turn has an adapter associated to it, which in turn can connect to the target resource to carry out the required operations.

An adapter provides the following benefits:

- It extends the internal logic and functionality of Oracle Identity Manager.
- It interfaces with any software resource, by connecting to that resource by using the API of the resource.
- It enables the integration between Oracle Identity Manager and an external system.
- It can be generated without manually writing code. However, Oracle Identity Manager does not restrict you from writing your own code for creating adapters.
- It is lightweight and specific to your needs.
- It can be maintained easily because all of the definitions for the adapter are stored in a repository. This repository can be edited through a GUI.
- One Oracle Identity Manager user can retain the domain knowledge about the integration, while another user can maintain the adapter.
- It can be modified and upgraded efficiently.

Adapters can be developed for a range of tasks:

- A process task adapter, which allows Oracle Identity Manager to automate the completion of a process task.
- A task assignment adapter, which enables Oracle Identity Manager to automate the assignment of a process task to a user or group.
- A rule generator, which incorporates business rules to the fields of either an Oracle Identity Manager form or a user-defined form (created by using the Form Designer form), so these fields can be populated automatically and saved to the Oracle Identity Manager database.
- A pre-populate adapter, which is a specific type of rule generator adapter that can be attached to a user-created form field. The data generated by this type of adapter can appear either automatically or manually. In addition, it uses criteria that enable Oracle Identity Manager to determine which pre-populate adapter will be applied to the designated form field. It populates the designated form field without saving this information to the Oracle Identity Manager database.
- An entity adapter, which is attached to an Oracle Identity Manager or user-created form field. Oracle Identity Manager triggers an entity adapter on preinsert, preupdate, predelete, postinsert, postupdate, or postdelete. After this occurs, the

field to which the adapter is attached is populated automatically and saved to the Oracle Identity Manager database.

Note: Oracle Identity Manager also allows you to create postprocessing handlers on entities, such as user, role, and organization.

3.2 Types of Adapters

This section provides additional details about the five adapter types.

Rule Generator Adapters

Certain business rules must be applied to perform field validations and enter default values into the forms which either come packaged with Oracle Identity Manager or are created by Oracle Identity Manager users. For example, for the Users form, you might want Oracle Identity Manager to generate the User ID automatically by concatenating the user's first name and last name.

To do this, you must create a specific type of adapter, which is designed to modify the field value in a form. This type of adapter, which can generate, modify, or verify the value of a form field automatically, is called a rule generator. Oracle Identity Manager triggers a rule generator on preinsert and preupdate.

After you create this adapter and attach it to a form, Oracle Identity Manager automatically updates the field value for all records of that form, and saves this information to the Oracle Identity Manager database.

If you create a rule generator that contains adapter variables, you must map these adapter variables to their proper locations. Otherwise, the adapter will not be functional.

You can also attach this type of adapter to a provisioning process. Once the process is provisioned to a target user or organization, Oracle Identity Manager will trigger the associated rule generator.

On occasion, a rule generator which has been assigned to a provisioning process might no longer be needed to complete the process. If this happens, you can remove the rule generator from the provisioning process. Similarly, after you attach one rule generator to a form field, you can connect a different rule generator to that form field. When this occurs, you must first remove the rule generator currently attached to the form field.

Entity Adapters

Similar to rule generator adapters, entity adapters are also responsible for generating, modifying, or verifying the value of a form field automatically, and saving this information to the Oracle Identity Manager database.

Note: In Oracle Identity Manager 11*g* Release 2 (11.1.2.3.0), creating new entity adapters and modifying existing entity adapters are not supported..

Some differences between rule generators and entity adapters are:

- **Execution schedule**. Entity adapters can be triggered by Oracle Identity Manager on preinsert, preupdate, predelete, postinsert, postupdate, and postdelete. A rule generator adapter can be executed only on preinsert and preupdate.

- **Manual field value modification**. The adapter populates the form field to which an entity adapter is attached. An Oracle Identity Manager user should not edit this value because the entity adapter will overwrite this modification. As a result, the modification will not be saved to the database.

  Similarly, the adapter also populates the form field to which a rule generator adapter is attached. However, an Oracle Identity Manager user can edit this value because this modification will take precedence over the value that the rule generator adapter generates. Because of this, the modification will be saved to the database.
- **Background color of form field**. If a rule generator is attached to a form field, the field will appear in a particular background color such as pink. This is a visual indicator that the field has a rule generator attached to it. On the other hand, when an entity adapter is attached to a form field, the field will not have a distinct background color.

Task Assignment Adapters

For a process task that must be completed manually, you can configure Oracle Identity Manager to automate the assignment of the task to either a specific user or a user who belongs to a particular role. This is achieved through the use of a task assignment adapter. Task assignment adapters are used only for assigning a task to a particular user or role.

When a task that is associated with specific provisioning process is created using the Tasks tab in the Process Definition form of the Design Console, you can choose the rule that decides if adapter will be picked up for execution. Note that this rule is defined in the Rule Definition form of the Design Console. An example of a rule is "Target User's Org name is XYZ. If this rule is satisfied, then the corresponding task assignment is picked up. However, you can have multiple rules defined and used while deciding task assignment. For multiple rules, Oracle Identity Manager associates priority with the task assignment functionality to decide the order in which the rule determination must occur. When the rule is determined, corresponding task assignment is run.

Note: In other words, the task assignment rule allows Oracle Identity Manager to decide whether to assign a process task to a user or role. The task assignment adapter enables Oracle Identity Manager to determine which user or role will be the recipient of the process task.

For this example, Oracle Identity Manager will trigger the Associate Adapter with User rule first (because it has the highest priority). If the condition of this rule is TRUE, it is successful. As a result, Oracle Identity Manager will associate the related task assignment adapter (the Assign Task to User adapter) with the process task.

On the other hand, when the condition of a rule is FALSE, the rule has failed. Oracle Identity Manager triggers the rule with the next highest priority. If this rule is successful, then Oracle Identity Manager assigns the designated adapter to the target process task.

So, in this example, if the Associate Adapter with User rule fails, then Oracle Identity Manager triggers the Associate Adapter with Role rule. If this rule is successful, then Oracle Identity Manager associates the related task assignment adapter (the Assign Task to Role adapter) to the process task.

After assigning a rule to a task assignment adapter, if this type of adapter contains adapter variables, you must map these variables to their proper locations. Otherwise, the adapter will not be functional.

Finally, when a task assignment adapter becomes invalid, or is no longer necessary for Oracle Identity Manager to allocate the process task to a user or group, you must remove the adapter from the task.

Prepopulate Adapters

Sometimes a user-created form contains both fields that can be populated by Oracle Identity Manager and fields into which an Oracle Identity Manager user must enter data. When the information that the user types into a field is contingent upon the data that appears in a system-generated field, Oracle Identity Manager must first populate this field. When the form is displayed, the user can view the system-generated data to enter information into the appropriate fields.

This is achieved by creating a type of rule generator known as a prepopulate adapter. By attaching it to a field designated to be system-generated, you enable Oracle Identity Manager to automatically populate this field with the appropriate information, without saving this information to the Oracle Identity Manager database.

The data generated by a prepopulate adapter can appear automatically or it can be manually entered. Oracle Identity Manager displays this information automatically when the Auto-prepopulate check box is selected for a provisioning process. When this check box is cleared, an Oracle Identity Manager user must manually generate the displaying of the data that is generated by the prepopulate adapter. To do this, click the prepopulate button on the form section of the Direct Provisioning wizard in the Web client, while provisioning the form to a user.

You can use the same prepopulate adapter for different form fields. In addition, you can designate multiple prepopulate adapters to be associated with a particular field. As a result, Oracle Identity Manager must know which prepopulate adapter it must select for the form field. This requires the use of prepopulate rules. These rules enable Oracle Identity Manager to select one prepopulate adapter, which is associated with a form field, when this prepopulate adapter is assigned to the field.

Each prepopulate adapter has a prepopulate rule associated with it. In addition every rule has a priority number which indicates the order in which Oracle Identity Manager triggers it.

For example, Oracle Identity Manager can trigger the Rule for Uppercase User ID rule first because it has the highest priority. If the condition of this rule is TRUE, it is successful. As a result, Oracle Identity Manager will attach the related prepopulate adapter (the Display Uppercase Letters for User ID adapter) to the User ID field.

On the other hand, when the condition of a rule is FALSE, the rule has failed. Oracle Identity Manager will trigger the rule with the next highest priority. If this rule is successful, Oracle Identity Manager will attach the associated adapter to the designated field.

So, in this example, if the Rule for Uppercase User ID rule fails, Oracle Identity Manager will trigger the Rule for Lowercase User ID rule. If this rule is successful, Oracle Identity Manager will attach the related prepopulate adapter (the Display Lowercase Letters for User ID adapter) to the User ID field.

After assigning a rule to a prepopulate adapter, if this type of adapter contains adapter variables, you must map these adapter variables to their proper locations. Otherwise, the adapter will not be functional.

Finally, when a prepopulate adapter associated with a field is no longer valid, you must remove the adapter from the field.

Process Task Adapters

A process task adapter enables Oracle Identity Manager to automatically execute process tasks in provisioning processes.

Each process and process task has a status, which indicates the stage of its completion. The statuses for a process or process task are listed in the following table in order of importance.

Task Status	Description
C	Completed: This process/process task has been completed successfully.
MC	Manually Completed: This process task has been completed successfully by an Oracle Identity Manager user (that is, manually).
P	Pending: This process/process task is in the process of being completed. All preceding tasks and processes, respectively, have been completed.
PX	Pending Cancellation: This process task will be canceled, but this task has to be completed first before it can be canceled.
R	Rejected: This process/process task has not been completed successfully or has not been approved. The status of rejected process tasks can only be changed to *Canceled* or *Unsuccessfully Completed*.
S	Suspended: This process/process task has been put on hold temporarily.
UC	Unsuccessfully Completed: This process task has been set to *Completed*. However, it had been rejected before.
W	Waiting: This process/process task cannot be completed until all preceding process tasks or processes are completed.
X	This process/process task has been stopped. Its status cannot change anymore

The status level of a process represents the most important status level of its process tasks, which must be completed for the process to be completed. Suppose a process has three process tasks, each process task has a different status level (*Completed*, *Waiting*, and *Rejected*), and all three process tasks must be completed for the process to complete. Because the highest task status level is *Rejected*, the status level of the process is also *Rejected*.

A process task can be managed in these ways:

- It can be handled manually by using the Object Process Console tab of the Organizations or Users forms, or the Oracle Identity Manager Web Application.
- An Oracle Identity Manager process can be configured so that one (or more) of its tasks is triggered automatically once it achieves a status of *Pending*.

3.3 Adapter Environment and Tools

This section contains these topics:

- Configuring the Adapter Environment
- The Adapter Factory
- Compiling Adapters

3.3.1 Configuring the Adapter Environment

To construct adapter tasks, ensure that Oracle Identity Manager has access to the target API JAR files and third-party applications to which you want to connect.

When your adapter uses Java tasks, you must configure Oracle Identity Manager to find the appropriate Java APIs. To do this, you must place the .jar files that contain these APIs into the Meta Data Store (MDS).

Then, you can access the Java classes associated with these Java APIs and use them in the Java task you are creating.

To configure Oracle Identity Manager to reference JAR and class files:

1. Open the JavaTasks subdirectory, which can be found within the `OIM_HOME/` directory path. For example, `C:\oracle\Xellerate\JavaTasks`.
2. Place the JAR file or files into this subdirectory. You can use these files to create Java tasks within an adapter without restarting the server.

Note:

When the Java code is in two different JAR files in the Adapter Factory, and in the adapter tasks if an object from the first JAR file (which has the common or shared code) is passed into the constructor of the next adapter task that is located in the second JAR file, then a compilation error is thrown.

As a workaround for this issue, ensure that the entire Java code is in a single JAR file only.

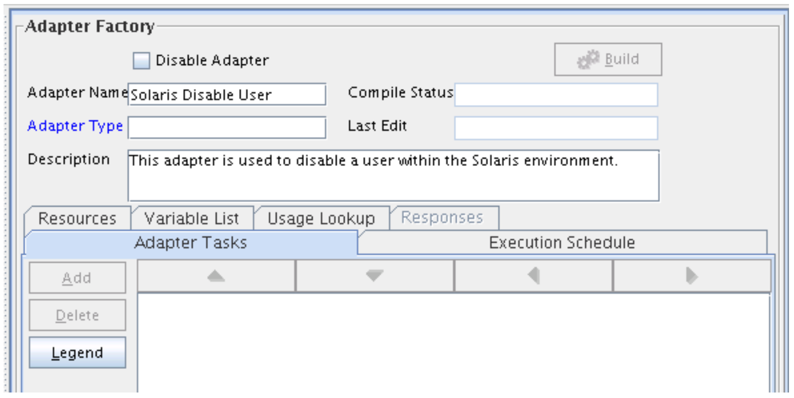
3.3.2 The Adapter Factory

As stated earlier, an adapter is a Java class created by an Oracle Identity Manager user through the Adapter Factory, which is accessed through the Design Console.

Adapters extend the internal logic and functionality of Oracle Identity Manager. In addition, they interact with any IT resource by connecting to that resource's API.

The Adapter Factory is a code-generation tool provided by Oracle Identity Manager that enables a user to create Java classes, known as adapters. Figure 3–1 shows the Adapter Factory Form in the Design Console.

Figure 3–1 Adapter Factory Form


Adapter Factory

Disable Adapter: ☐

Adapter Name: Solaris Disable User

Compile Status:

Adapter Type:

Last Edit:

Description: This adapter is used to disable a user within the Solaris environment.

3.3.3 Compiling Adapters

Oracle Identity Manager provides various options for compilation, including:

- compile individual adapters one at a time
- compile a set of adapters at once
- compile all adapters that exist in the Oracle Identity Manager database with a single click

3.3.3.1 Automatic Compilation of Adapters

Adapters are compiled automatically when you import connector files by using the Deployment Manager. The compiled adapter class files are stored in the Oracle Identity Manager database, as opposed to the file system, from where they are loaded at run time. The following two APIs are available to compile adapters programmatically:

- `public void compileAdapter (String adapterName)`: This API compiles a single adapter and stores the compiled classfile in the database. It takes the name of the adapter as a parameter. If the adapter is not found or if there are any errors, the API throws an appropriate exception.
- `public void compileAll`: This API compiles all adapters in a system. If it encounters any errors during compilation, it throws an exception of the type `tcBulkException`. This exception comprises all the individual errors that the API encounters during compilation.

You can modify the adapters manually if you make any changes.

Note: You must set the path of the JDK directory in the `XL.CompilerPath` system property. Otherwise, an error is encountered during the adapter compilation stage when you import an XML file using the Deployment Manager.

Refer to the "System Properties in Oracle Identity Manager" in the *Oracle Fusion Middleware System Administrator's Guide for Oracle Identity Manager* for information about setting values of system properties.

3.3.3.2 Compiling Adapters Manually

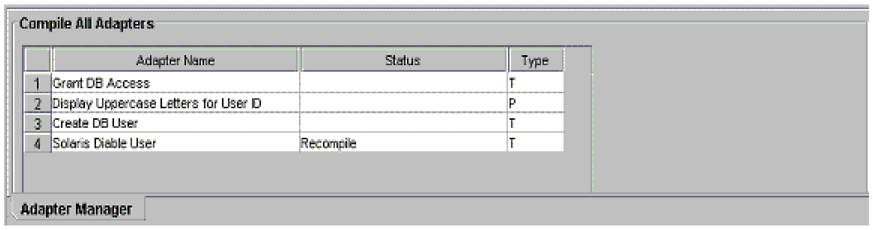
The Adapter Manager form is located in the Development Tools folder. You use it to compile multiple adapters simultaneously.

To manually compile multiple adapters, perform these steps:

1. Open the Adapter Manager form.

   The Adapter Manager form is in the Development Tools folder. It is used to compile multiple adapters simultaneously, as shown in Figure 3–2.

Figure 3–2 Adapter Manager Form



	Adapter Name	Status	Type
1	Grant DB Access		T
2	Display Uppercase Letters for User ID		P
3	Create DB User		T
4	Solaris Diable User	Recompile	T

2. To compile every adapter that resides within the Oracle Identity Manager database, select the **Compile All** option.

   To compile multiple adapters, select the adapters you want to compile. Then, select the **Compile Selected** option.

   To compile all adapters that do not have an OK status, select the **Compile Previously Failed** option.

3. Click the **Start** button.

   Oracle Identity Manager will compile the adapters that match the criteria you specified in Step 2.

   > **Tip:** Oracle Identity Manager lets you review the record of any adapter that appears within the Adapter Manager form to see detailed information about the adapter.
   >
   > To view an adapter's record, select the desired adapter and either double-click its row header, or right-click the adapter, and select the Launch Adapter command from the menu that appears.

3.4 Defining Adapters

To define an adapter:

1. Log in to Oracle Identity Manager Design Console.
2. Open the Adapter Factory form. This form is in the Development Tools folder in the Design Console.
3. In the Adapter Name field, enter the name of the adapter, for example, `Create Solaris User`.

   > **Note:** Although the adapter name can contain special characters, Oracle recommends that you do not use them because there might be run-time errors.

4. Double-click the Adapter Type lookup field.

   The Lookup window is displayed, displaying the five types of Oracle Identity Manager adapters. These are:

   - Process Task
   - Rule Generator
   - Pre-populate Rule Generator
   - Entity

- Task Assignment

5. To enable the adapter to automate a process task, select **Process Task (T)**.

   To incorporate business rules into an Oracle Identity Manager or user-defined form field, select **Rule Generator (R)**. For example, for the User ID field of a form, you can configure Oracle Identity Manager to concatenate the initial letter of the user's first name with the user's last name.

   You can attach a type of rule generator adapter to a user-created form field, so that it can:

   - Display the data, which is generated by the adapter, automatically or manually.
   - Use criteria that enable Oracle Identity Manager to determine which adapter is applied to the designated form field.

   To attach the adapter to an Oracle Identity Manager or user-defined form field, and have Oracle Identity Manager trigger the adapter on preinsert, preupdate, predelete, postinsert, postupdate, or postdelete, select **Entity (E)**.

   To allow the adapter to automate the allocation of a process task to a user or group, select **Task Assignment (A)**.

   > **Tip:** If you create an entity adapter, then an error might be generated while compiling the adapter on computers with less file limits. To avoid this problem, change the file limits in the /etc/security/limits.conf file to the following:
   >
   > soft nofile 4096
   >
   > hard nofile 4096
   >
   > Then, restart Oracle Identity Manager.

6. Select the type of adapter you want, for example, Process Task (T). Then, click **OK**.
7. In the Description field, type a description for the adapter, for example, `This adapter is used to create a new user for the Solaris environment`.
8. From the toolbar, click **Save**.

   The adapter is now stored in the Oracle Identity Manager database.

3.5 Tabs of the Adapter Factory Form

The Adapter Factory form in the Design Console contains the following tabs:

- Adapter Tasks
- Resources
- Variable List
- Usage Lookup
- Responses

3.5.1 Adapter Tasks

In the Adapter Tasks tab, you can create and manage the atomic function calls of an adapter. These function calls are known as adapter tasks.

The sequence of calls is vital because these calls in turn gets converted into Java statements. In other words, if you put an Else call before an If call, then the adapter is not compiled. In addition, you must understand the logical flow of java program while creating adapter. Analogically, this is like writing an algorithm instead of a program with Java syntax.

3.5.2 Resources

From the Resources tab, you can:

- Click the Java APIs subtab to see the Java APIs that are being used by the adapter.
- Click the Other subtab to document a non-Java API file to the adapter, if necessary.

Note: This Resources tab does not represent resource objects.

3.5.3 Variable List

For prepopulation adapters, the data is passed to adapter input variables and are processed by using adapter logic. The adapter returns output variable, which is then assigned to process form field.

From the Variable List tab, you can:

- Create, modify, and delete adapter variables.
- Set the data type and provide a description for each variable.
- Map an adapter variable to a literal or an adapter reference. You can also postpone the mapping until it is attached to a process task or a form field.

You also can resolve the value of the adapter variable at run time, when it is attached to a process task and the process task is run. As a result, process-specific data is available to map to this variable.

3.5.4 Usage Lookup

For a process task or task assignment adapter, the Usage Lookup tab displays the process task to which the adapter is attached, as well as the process of which this process task is a member.

For a rule generator or entity adapter, this tab shows the Oracle Identity Manager form and associated data object to which the adapter is attached. In addition, it displays the execution schedule of the adapter, along with a sequence number that represents the order in which Oracle Identity Manager will trigger the adapter.

For a pre-populate adapter, this tab displays the user-defined form and form field to which the adapter is attached. Also, it shows the pre-populate rule that is associated with the adapter.

3.5.5 Responses

The Responses tab is used for defining meaningful responses to the process task. These responses depend on the execution result of the adapter. The various error messages returned by the external system can be mapped to these responses in a way that they make sense in the context of the process task. On attaching the adapter to a process task, the status bucket, which consists of Pending, Completed, and Rejected, of the process task (and subsequently the Object status) can be set, based on the adapter response code.

Tip: Oracle Identity Manager enables the Responses tab only for process task adapters. If an adapter is a task assignment, rule generator, pre-populate, or entity adapter, Oracle Identity Manager disables this tab.

3.6 Disabling and Re-enabling Adapters

To disable an adapter so that it cannot be used with a process task or form field, select the **Disable Adapter** option, and save the adapter.

To re-enable it, clear the **Disable Adapter** option, and save the adapter.

3.7 About Adapter Variables

For a newly-created adapter to work, you can map data to the parameters of the adapter tasks. For this reason, you create placeholders, also known as adapter variables, to map the data at run time.

Note: An adapter variable can be reused for all adapter tasks.

Once an adapter variable is not needed for the adapter to run, you can remove it from the adapter. After you have deleted the adapter variable, ensure to recompile the adapter.

3.7.1 Creating an Adapter Variable

To create an adapter variable:

1. Select the adapter to which you wish to add an adapter variable, for example, the `Create Solaris User` adapter.
2. Select the Variable List tab.
3. Click **Add**.

   The Add a Variable window is displayed.
4. When you do not want Oracle Identity Manager to be able to change the adapter variable value after it is activated, select **Final**.
5. In the Variable Name field, enter the name of the adapter variable, for example, `SolarisUserID`.

   Caution: The adapter variable name cannot contain spaces.

6. From the Type menu, select the classification type of the adapter variable, such as String. The available items are:
   - Object
   - IT Resource
   - String
   - Boolean
   - Character
   - Byte

- Date
- Integer
- Float
- Long
- Short
- Double

7. Within the Description text area, you can enter explanatory information about the adapter variable.
8. From the Map To menu, you can map your adapter variable to one of the items listed in Table 3–1.

Table 3–1 Items on the Map To Menu

Name	Description
Literal	This adapter variable is mapped to a constant (or literal).
Resolve at Run time	This adapter variable's mapping occurs later, at run time. Selecting this option increases the reusability of the adapter.
Adapter References	This adapter variable gives access to an Oracle Identity Manager database reference or an Oracle Identity Manager data object reference.
System Date	When this adapter variable is triggered by Oracle Identity Manager, it is mapped to the current date and time of the Server. **Note**: This option appears only when you select the Date type.

Note: When you select the object type, a Qualifier menu is displayed within the Add a Variable window. From this menu, you can select either of the following:

- Database Reference. If you select this item, the adapter variable is mapped to the reference of the database that the Oracle Identity Manager is currently running against.
- Data Object Reference. If you select this item, the adapter variable is mapped to an Oracle Identity Manager data object.

Note: If you select the IT Resource type, a Resource Type menu is displayed within the Add a Variable window. From this menu, you can select one of the IT resource types that have been created by using the IT Resource Type Definition form. By doing so, you can map the adapter variable to a parameter of this IT resource type.

9. On the toolbar in the Add a Variable window, click **Save**. The information for your adapter variable is stored in the Oracle Identity Manager database.

   Close the Add a Variable window to activate the main screen. The name, classification type, mapping selection, and description of the adapter variable you created appear in the child table of the Variable List tab.

This adapter variable now belongs to the adapter in the Adapter Factory form. It is saved to the Oracle Identity Manager database, and the adapter variable is ready to use.

3.7.2 Modifying an Adapter Variable

To modify an adapter variable:

1. Select the adapter that contains the adapter variable you want to edit, for example, the `Create Solaris User` adapter.
2. Click the Variable List tab and double-click the row header of the adapter variable you want to modify. The Edit a Variable window is displayed, showing information about the adapter variable.
3. Make the necessary edits, for example, changing the adapter variable's data type from String to Character.
4. On the Edit a Variable toolbar, click **Save**. The modified information about the adapter variable is stored in the Oracle Identity Manager database.
5. Close the Edit a Variable window to activate the main screen. The adapter variable you modified appears within the child table of the Adapter Factory form.

Note: Ensure that you check your data mappings and recompile the adapter, especially if you change the adapter variable's data type.

3.7.3 Deleting an Adapter Variable

When an adapter variable is no longer necessary for the adapter to run, you can remove it from the adapter. To do this:

1. Select the adapter that contains an adapter variable you want to remove, for example, the `Create Solaris User` adapter.
2. Select the Variable List tab.
3. From the list of this tab, select the adapter variable you want to delete.
4. Click **Delete**.
5. Recompile the adapter after deleting any variable.

The adapter variable disappears from the child table. The adapter variable has been deleted.

3.8 Creating Adapter Tasks

After you construct the adapter and create its variables, you can create the atomic function calls of an adapter. These function calls are known as adapter tasks.

This section explains adapter tasks and how to create tasks:

- Types of Adapter Tasks
- Creating a Java Task
- Reassigning the Value of an Adapter Variable

3.8.1 Types of Adapter Tasks

Oracle Identity Manager allows you to create the following adapter tasks:

- A Java task, which allows an adapter to communicate with an external source by invoking Java API.
- A utility task, which enables you to populate an adapter with methods and APIs that come packaged with Oracle Identity Manager. In addition, this type of task provides you with access to the Java Standard Library APIs.
- An Oracle Identity Manager API task, which enables access to Oracle Identity Manager published APIs from adapter tasks. This allows for enhanced portability of adapter code.
- A set variable task, which allows you to set a variable within an adapter.
- An error handler task, which lets you display any errors associated with an adapter that occur at run time. In addition, you can see the reasons for the errors, along with possible solutions.
- A logic task, which lets you build a conditional statement within an adapter.

You can create the following types of logic tasks:

- FOR loops
- WHILE loops
- IF statements
- ELSE statements
- ELSE IF statements
- BREAK statements
- RETURN statements
- CONTINUE statements
- SET VARIABLE statements
- Handle Error statements

For classification purposes, Oracle Identity Manager represents each type of adapter task by an icon. The icon, which precedes the task name, is a visual indicator of the type of task it is. For example, "J" represents a Java task, and "LT" represents a logic task.

To see a list of these icons, select the Adapter Tasks tab, and click **Legend**. The Legend window appears, displaying the following list of icons:

- Functional Task
  - Java
- Utility Task
  - Utility
  - Oracle Identity Manager API
- Logical Task

3.8.2 Creating a Java Task

Oracle Identity Manager can handshake with an external source through a Java API. To make this happen, you must add a task to an adapter which, when triggered by Oracle Identity Manager, initiates communications with the external source. This type of task is called a Java task.

To create a Java task:

1. Select the adapter to which you want to add a Java task, for example, the Update Solaris Password adapter.
2. Select the Adapter Tasks tab.
3. Click **Add**.

   After the Adapter Task Selection window is displayed, select the **Functional Task** option.
4. From the display area to the right of this option, select the Java item, and click **Continue**.

   The Object Instance Selection window is displayed.

   Table 3–2 explains the options in the Object Instance Selection window.

Table 3–2 Options in the Object Instance Selection Window

Option	**Description**
New Object Instance	When you click this option, you are creating a new Java object instance.
Persistent Instance	You can call the method on a persistent object by clicking this option, clicking the adjacent combo box, and selecting an object instance from the drop-down menu.
Task Return Value Instance	You can call this method on an object returned by an adapter task defined earlier by clicking this option, clicking the combo box, and selecting an adapter task from the drop down list.

> **Note:** When the Persistent Instance option is grayed out, it indicates that you have not defined any persistent objects for your adapter. Similarly, if the Task Return Value Instance option is grayed out, none of the tasks have Java Object return values associated with them.

5. Click an option—for example, New Object Instance—and click **Continue**. The Add an Adapter Factory Task window is displayed.

   Table 3–3 lists and describes the various regions of the Add an Adapter Factory Task window:

Table 3–3 Regions of the Add an Adapter Factory Task Window

Name	**Description**
Task Name	This field displays the name of the Java task.
Persistent Instance	If this Java object is to be used again, the check box is selected, and the name of the task instance is entered in the adjacent field.
API Source	This combo box contains a list of all JAR and class files to which you have access.
Application API	This combo box contains a list of all class files to which you have access, and which belong to the JAR file that has been selected from the API Source list.
Constructors	This text area displays all the constructors, which are available for the Java object.

Table 3–3 (Cont.) Regions of the Add an Adapter Factory Task Window

Name	Description
Methods	This text area shows a list of all the methods, which are available for the Java object.
Application Method Parameters	This area contains the parameters of the selected constructor and method. These parameters are mapped to the adapter variables and Oracle Identity Manager components.

6. In the Task Name field, enter the name of the task you are creating, for example, `Update Password`.
7. (Optional.) To make your Java object reusable, select **Persistent Instance**, type the name of the instance of this task in the text field located to the right of the check box.

   Caution: Ensure that name of the instance contains no spaces.

   Note: To reference a session with the target resource multiple times during the life of the adapter, and not just once, select **Persistent Instance**.

   Tip: By setting the Java object to be persistent, the next time you create a Java object, it appears in the Persistent Instance list of the Object Instance Selection window. In addition, you do not have to map the constructor to all adapter tasks of the same Java object.

8. Select the API Source. The JAR files appear, which Oracle Identity Manager references from the JavaTasks subdirectory of the *`OIM_HOME`*`/` directory path—for example, `C:\oracle\Xellerate\JavaTasks`.

   See Also: Section 3.3.1, "Configuring the Adapter Environment" for instructions on how to enable Oracle Identity Manager to use third-party JAR files with a Java task

9. Select the Application API. The class files, which belong to the JAR file you selected in the API Source, appear.
10. From the Constructors area, select the method to be used to initialize the Java class you selected.
11. From the Methods area, select the method that will be used with your Java task.
12. From the toolbar, click **Save**.

    The information pertaining to the Java task is stored in the Oracle Identity Manager database. You can now access the parameters of your Java task's constructors and methods. These parameters appear in the Application Method Parameters region of the Add an Adapter Factory Task window.

13. To display the Java class constructors and methods for which you must set mappings, click the plus icons displayed to the left of the Constructor and Method icons.

14. Select the parameter of the constructor or method for which you must set a mapping.
15. In the Description text area, you can enter a description for this mapping.
16. Click the Map to combo box, and select an item that you can map to the parameter of the constructor or method, for example, Adapter Variables.
17. Set the appropriate mappings.

    See Also: "Adapter Mapping Information" on page 3-34 for more information about which mappings to set

18. Click **Set**.

    The parameter of the selected constructor or method now appears in blue. This signifies that it has been mapped.

    Tip: To remove a parameter mapping, right-click the appropriate parameter, and select Un-Map Parameter from the popup menu that appears.

19. Repeat steps 15 through 18 for all parameters of the constructors and methods that appear in the Application Method Parameters region.
20. On the Add an Adapter Factory Task window toolbar, click **Save**. The information pertaining to the Java task is stored in the Oracle Identity Manager database.
21. On the toolbar, click **Close**. The Add an Adapter Factory Task window disappears, and the main screen is active once again. The Java task that you created—for example, Update Password—appears within the Adapter Factory form.
22. (Optional.) To create additional Java tasks for the adapter, repeat steps 3-21.

    Tip: You can create different types of adapter tasks, and add them to the adapter.

    If the adapter is logically complete, and all variables on the adapter tasks are mapped, you can compile it to use with a process task or form field.

23. To compile the adapter, click **Build**.

    The text in the Compile Status field changes from Recompile to OK. This indicates that Oracle Identity Manager compiled the adapter and found no errors. You can now attach the adapter to a process task or form field.

24. (Optional.) To see the code that Oracle Identity Manager generates, from the toolbar, click **Notes**.

    The Notes window is displayed, containing the code that Oracle Identity Manager generated.

Note: If, after clicking Build, CODE GEN ERROR appears in the Compile Status field, it means that Oracle Identity Manager encountered one of two types of errors while validating and compiling the adapter:

- Validation Error

  While Oracle Identity Manager is checking the adapter to verify that it is valid, an error is found. This error can result from a parameter of an adapter task not being mapped, a parameter being mapped improperly, or an adapter task being placed out of order.

  Because Oracle Identity Manager generates code for an adapter only after it is validated, if Oracle Identity Manager encounters a validation error, it does not create any code.

- Java Compilation Error

  Oracle Identity Manager has verified that the adapter is valid. However, while Oracle Identity Manager is compiling the adapter, an error is found. This error can result from assigning an incorrect data type to an adapter task parameter.

  Because Oracle Identity Manager has validated the adapter, it generates code. However, Oracle Identity Manager stops building code at the point of the compilation where it encounters the error.

Tip: Once you create a Java task, and add it to an adapter, you can see the following information by accessing the Resources tab of the Adapter Factory form:

- The JAR and class files used to create the Java task.
- The name, which represents the directory path that contains these JAR and class files.

3.8.3 Reassigning the Value of an Adapter Variable

Sometimes, for an adapter to accomplish its required objective, you must reassign the value of one adapter variable to another adapter variable, a different type of adapter task, or a constant (or literal). The task that enables you to reallocate an adapter variable value is known as a set variable task.

See Also: "About Adapter Variables" on page 3-12 for information about adapter variables

For example, you can create a set variable task to set the adapter variable return value to equal the output of an adapter task (UserName) if the User ID length is fewer than 11 characters.

To create a set variable task:

1. Select the adapter to which you wish to add a set variable task (for example, the Check the Solaris User ID adapter).
2. Click the Adapter Tasks tab.
3. Click **Add**. The Adapter Task Selection window is displayed.

4. Select the Logic Task option.
5. From the display area, select SET VARIABLE, and click **Continue**. The Add Set Variable Task Parameters window is displayed.
6. From the Variable Name list, select the adapter variable that has a value you want to reassign—for example, Adapter return value.
7. From the Operand Type list, select the type of operand that will provide the value for the variable.

   Tip: You can reassign an adapter variable's value to another adapter variable, a different type of adapter task, or a literal.

   Use Table 3–4 to understand the various types of operands.

Table 3–4 Types of Operands

Operand Name	Description
Variable	If you select this operand type, adapter variables appear in the Operand Qualifier list. From this list, select the specific adapter variable that will provide the reassigned value. **Note**: The only adapter variables that will appear in the Operand Qualifier combo box will be those variables that have the same data type as the adapter variable that is displayed within the Variable Name combo box.
Adapter Task	By selecting this operand type, adapter tasks are displayed in the Operand Qualifier combo box. From this combo box, select the particular adapter task that will provide the reallocated value. **Note**: The only adapter tasks that will appear in the Operand Qualifier combo box will be those tasks that have the same data type as the adapter variable that is displayed within the Variable Name combo box.
Literal	When you select this operand type, types of literals appear in the Operand Qualifier combo box. From this combo box, select the type of literal that will provide the reallocated value. Then, type the specific literal into the field that appears underneath the combo box.

The following task sets the adapter variable's return value to be equal to the UserName adapter variable.

1. On the toolbar in the Add Set Variable Task Parameters window, click **Save**. The set variable task you created is stored in the Oracle Identity Manager database.
2. On the Add Set Variable Task Parameters window toolbar, click **Close**. The Add Set Variable Task Parameters window disappears, and the main screen is active once again. The set variable task that you created, for example, `Set Adapter return value = UserName`, appears in the Adapter Factory form.
3. (Optional.) Repeat Steps 3-9 to create additional set variable tasks for the adapter.

   You are now ready to compile the adapter, so it can be used with a process task or form field.
4. To compile the adapter, click **Build**. The text in the Compile Status field changes from Recompile to OK. Oracle Identity Manager compiled the adapter and found no errors. You can attach the adapter to a process task or form field.

3.9 Modifying Adapter Tasks

The following procedure will show you how to edit an adapter task, in case you must make changes to it. To modify an adapter task

1. Select the adapter that contains the adapter task you wish to edit (for example, the *Update Solaris User Group* adapter).
2. Click the **Adapter Tasks** tab.
3. Double-click the adapter task that you want to modify.

   The Edit Adapter Factory Task Parameters window is displayed, displaying information that relates to the adapter task you selected. Within this window, make the necessary modifications.
4. On the Edit Adapter Factory Task Parameters window toolbar, click **Save**.

   The information you modified is stored in the Oracle Identity Manager database.
5. On the toolbar, click **Close**.

   The Edit Adapter Factory Task Parameters window disappears. The main screen is active again. The modified task appears within the child table of the **Adapter Factory** form. You must re-compile the adapter, so it can be used with a process task or form field.
6. To recompile the adapter, click **Build**.

   The text in the **Compile Status** field changes from *Recompile* to *OK*. This indicates that Oracle Identity Manager compiled the adapter and did not find any errors. You can now attach the adapter to a process task or form field.

Caution: You cannot modify the API call inside a Java, Xellerate API, or Utility task. The adapter task has to be deleted and re-created.

In addition, if *CODE GEN ERROR* appears in the Compile Status field, Oracle Identity Manager encountered errors while compiling the adapter. Rectify the errors, if necessary re-do the adapter task modifications, and compile the adapter again.

3.10 Changing the Order and Nesting of Tasks

If you add multiple tasks to an adapter, you can either change the order in which the tasks are executed, or place one task inside of another task for the adapter to work.

The following procedure will show you how to change the order and nesting of tasks.

Caution: You should not change the order and nesting of adapter tasks unless you understand the mapping dependencies of the adapter tasks.

To change the order and nesting of tasks:

1. Select the adapter that contains tasks of which you want to change the order and/or nest (for example, the *Check the Solaris User ID* adapter).
2. Click the **Adapter Tasks** tab.

   The tasks appear, which belong to the current adapter.

In this example, the following changes must occur:

- The error handler task must be nested inside of the *IF (Check ID Length > 10)* logic task.
- The set variable task has to be nested inside of the **ELSE** logic task.
- The **IF** logic task precedes the **ELSE** logic task.

Therefore, you must first reorganize the logic tasks. Then, you must nest the error handler task and set variable task inside of the **IF** and **ELSE** logic tasks, respectively. To reorganize tasks:

3. Select the task that must run before another task, and click the **Up** arrow button. The selected task will switch places with the task that precedes it.

   or

   Select the task that must be executed after another task, and click the **Down** arrow button. The highlighted task is displayed below the task that previously followed it.

   To nest tasks/remove task nestings:

4. Select the task that must be placed inside of another task, and click the **Right** arrow button. The selected task will be nested inside of the task that appears above it.

   or

   Select the task that no longer be nested inside of another task, and click the **Left** arrow button. The highlighted task will not be nested inside of the task that is displayed above it.

5. On the toolbar, click **Save**.

   The order and nesting of the adapter's tasks is stored in the Oracle Identity Manager database. If the adapter is logically complete and all variables on the adapter tasks are mapped, you can compile it to use with a process task or form field.

6. To compile the adapter, click **Build**.

   The text in the Compile Status field changes from *Recompile* to *OK*. This indicates that Oracle Identity Manager compiled the adapter and did not find any errors. You can now attach the adapter to a process task or form field.

   Caution: If you see *CODE GEN ERROR* in the Compile Status field, Oracle Identity Manager found errors while compiling the adapter. Rectify the errors, if necessary re-do the adapter task modifications, and compile the adapter again.

3.11 Deleting Adapter Tasks

When an adapter task is no longer necessary for the adapter to run, you must remove it from the adapter. To delete an adapter task:

1. Select the adapter that contains the task you wish to remove (for example, the *Update Solaris User Group* adapter).
2. Click the **Adapter Tasks** tab.
3. Select the task that you want to remove (for example, the *CONTINUE* logic task).

4. Click **Delete**.

   The selected task is deleted and disappears from the child table.

5. On the toolbar, click **Save**.
6. Recompile the adapter.

> **Caution:** While deleting adapter tasks, ensure that the logic of the adapter is consistent and maintained.

3.12 Working with Responses

Adapters can have different outcomes, called **responses**. Based on these responses, adapters can trigger other process tasks.

For example, if the adapter returns a *True* response, the process task's status can be set automatically to *Completed*. However, if the adapter returns a *False* response, the process task's status can be set automatically to *Rejected*, and another process task can be triggered.

These responses can be added, modified, or removed on the Responses tab of the Adapter Factory form.

The following procedures will show you how to create, modify, and delete responses.

> **Note:** Responses are used only with process task adapters, because these adapters are attached to process tasks. Rule generators, pre-populate adapters, and entity adapters are not connected to processes. In addition, task assignment adapters are not associated with responses. Therefore, if the active adapter is a task assignment adapter, rule generator, pre-populate adapter, or entity adapter, Oracle Identity Manager disables the Responses tab.

3.12.1 To Create a Response

1. Select the adapter to which you want to add responses (for example, the *Create Solaris User* adapter).
2. Click the **Responses** tab.
3. Click **Add**.

   An empty row is inserted into the **Responses** tab.

4. Click the field that appears within the **Code Name** column.
5. Enter a code, which represents a response type that can be generated (for example, *True*).
6. Click the field that appears within the **Description** column.
7. Enter a description for this response (for example, *The user was created successfully.*).
8. Double-click the field that appears within the **Status** column.

   The Lookup popup window is displayed, containing the different status levels that you can associate with the response.

Note: For more information about Oracle Identity Manager's status levels, refer to Chapter 4, "About Process Task Adapters" on page 4-1.

9. Click the desired status level (for example, *Completed (C)*). Then, click **OK**.

   The Lookup window disappears, and the **Responses** tab is active once again.

10. Create another response, by clicking the **Add** button, and entering *False* and *The user was not created successfully.* into the Code Name and Description fields, respectively. Then, access the Lookup window, and assign the *Rejected (R)* status level to this response.

11. On the toolbar, click **Save**.

    The responses that you created for this adapter have been stored in the Oracle Identity Manager database. After you attach this adapter to a process task, these responses will appear in the Responses tab of the Editing Task window of the Process Definition form.

3.12.2 To Modify a Response

The following procedure demonstrates how to edit a response.

1. Select the adapter that contains the response you want to edit (for example, the *Create Solaris User* adapter).
2. Click the **Responses** tab.
3. Double-click the field of the response, which contains information that you want to modify.

   a. If the field is a text field, Oracle Identity Manager enables it. You can now edit the contents within this field.

   b. When the field is a lookup field, the Lookup popup window is displayed, containing the different status levels that you can associate with the response. Click the desired **status level**, click **OK**.

   For example, double-click the **Status** column of the *False* response, select the *Suspended (S)* status level, and click **OK**.

4. On the toolbar, click **Save**.

   The information that you modified for the response is stored in the Oracle Identity Manager database.

3.12.3 To Delete a Response

When a response is no longer necessary, you can delete it from the adapter.

1. Select the adapter, which contains a response that you want to remove.
2. Click the **Responses** tab.
3. Select the response that you want to delete.
4. Click **Delete**.

The response disappears. This indicates that Oracle Identity Manager has deleted the response.

3.13 Working with Prepopulate Adapters

This section contains these topics:

- Attaching Prepopulate Adapters to Form Fields
- Removing Prepopulate Adapters from Form Fields

3.13.1 Attaching Prepopulate Adapters to Form Fields

To attach a prepopulate adapter to a form field, perform the following steps:

1. Select the field to which a prepopulate adapter will be attached.
2. Select the rule that will determine if the adapter will be used to populate the designated field with information.
3. Select the adapter that will be associated with the designated field.
4. Set the priority number of the selected rule.
5. Map the adapter variables of the prepopulate adapter to their proper locations.

Note: To attach a prepopulate adapter to a form field, you must ensure the following:

- The form is not in an active state. Otherwise, create a new form version.
- After attaching the adapter, you must activate the form to be able to use it.

6. Open the Form Designer form.
7. Query for the form to which you want to attach a prepopulate adapter (for example, Solaris).
8. Click the **prepopulate** tab.

   The prepopulate adapters, which have already been attached to the form you queried, appear within this tab.

Note: If no adapters have been attached to a form field, the prepopulate tab will be empty.

If a process form has two IT resource fields, then the second IT resource must be populated using programmatic mechanism and prepopulate adapters. Two IT resources cannot be populated because the UI Form Designer does not support an IT resource type widget.

9. Click **Add**.

   The prepopulate Adapters dialog box is displayed.

   Table 3–5 lists and describes the fields of the prepopulate Adapters dialog box.

Table 3–5 Fields of the Prepopulate Adapter Dialog Box

Name	Description
Field Name	This combo box contains a list of all of the form fields to which a prepopulate adapter can be attached.

Table 3–5 (Cont.) Fields of the Prepopulate Adapter Dialog Box

Name	Description
Rule	From this lookup field, select the rule that will determine if the associated adapter will be used to populate the designated form field with information.
Adapter	From this lookup field, select the adapter that will be associated with the designated field.
Order	From this field, set the priority number of the selected rule.
Adapter Status	This field displays the mapping status of the adapter variables. See "Attaching Process Task Adapters to Process Tasks" on page 3-29 for information about the various mapping statuses for an adapter.
Adapter Variables	This area displays the following: ■ Mapped: The mapping statuses of the adapter's variables. "Y" indicates that an adapter variable has been mapped properly; "N" indicates that this variable has not been mapped correctly. ■ Name: The names of the adapter variables. ■ Mapped to: The form fields to which the variables are mapped If an adapter variable is not yet mapped, the corresponding cell in this column will be empty.

10. From the **Field Name** combo box, select the form field, such as User ID, to which the prepopulate adapter will be attached.
11. Double-click the **Rule** lookup field. From the Lookup dialog box that is displayed, select the rule that will determine if the associated adapter will be used to populate the designated form field with information (for example, Rule for Lowercase User ID).
12. Double-click the **Adapter** lookup field. From the Lookup dialog box that is displayed, choose the adapter that will be associated with the field you selected in Step 10, for example, Display Lowercase Letters for User ID.
13. In the **Order** field, enter the priority number of the rule you selected in Step 11, for example, 2.
14. On the prepopulate Adapters window toolbar, click **Save**.
15. Mapping Incomplete appears within the Adapter Status field. This signifies that the adapter you selected contains variables that have not been mapped correctly. These variables can be mapped to their proper locations. Otherwise, the adapter will not work.
16. Set the mappings for each variable that appears in the Adapter Variables region of the prepopulate Adapters window. To do so, double-click the row header of the variable you want to map, for example, UserID.

    The Map Adapter Variables window is displayed.

    Table 3–6 describes the fields of the Map Adapter Variables window.

Table 3–6 Fields of the Map Adapter Variables WIndow

Field Name	Description
Variable Name	This field displays the name of the adapter variable for which you are setting a mapping (for example, UserID).

Table 3–6 (Cont.) Fields of the Map Adapter Variables WIndow

Field Name	Description
Data Type	This field shows the data type of the adapter variable (for example, *String* is the data type for the UserID adapter variable).
Map To	This field contains the types of mappings that you can set for the adapter variable (for example, Process Data). When you map the adapter variable to a location or a contact, Oracle Identity Manager enables the adjacent combo box. From this combo box, select the specific type of location or contact to which you are mapping the adapter variable. If you are not mapping the adapter variable to a location or contact, this combo box is grayed out.
Qualifier	This field contains the qualifiers for the mapping you selected in the **Map to** combo box (for example, User ID).
IT Asset Type	This field enables you to select a specific IT Resource (for example, Solaris) when you map an adapter variable to an IT Resource, and this variable's data type is String. If you are not mapping the adapter variable to an IT Resource, or the variable's data type is not String, this field does not appear.
IT Asset Property	This field enables you to select a specific field that will receive the results of the mapping (for example, *User Name*), when you map an adapter variable to an IT Resource, and this variable's data type is String. If you are not mapping the adapter variable to an IT Resource, or the variable's data type is not String, this field does not appear. **Important**: The IT Asset Type and IT Asset Property fields are included within this window for backward compatibility. The preferred way is to create an adapter variable with a data type of IT Resource, in which case these fields will not appear.
Literal Value	When you map the adapter variable to a literal, use this field to specify the specific literal value. If you are not mapping the adapter variable to a literal, this field does not appear.

17. Complete the Map To, Qualifier, IT Asset Type, IT Asset Property, and Literal Value fields.

    See Also: "Adapter Mapping Information" on page 3-34 for more information about the mappings to select

18. On the Map Adapter Variable window toolbar, click **Save**. Then, click **Close**.

    The Map Adapter Variables window disappears. The prepopulate Adapters window is active again.

    The text in the Adapter Status field changes from Mapping Incomplete to Ready. In addition, the mapping statuses for the adapter's variables change from No (N) to Yes (Y).

19. On the prepopulate Adapters window toolbar, click **Close**.

The prepopulate Adapters window disappears, and the Form Designer form is active again. The prepopulate adapter, which you attached to the User ID form field (Display Lowercase Letters for User ID), appears in the prepopulate tab of the Results of 1Q Sales 2003 form.

After a process, which references this form, is provisioned to a target user or organization, the form will appear. Oracle Identity Manager will check to see if the prepopulate rule, which has the highest priority, is valid. If so, Oracle Identity Manager will assign the associated prepopulate adapter to the designated field (User ID), and execute it. At this point, one of the following actions occur:

- If the Auto-prepopulate check box is selected for the provisioning process, Oracle Identity Manager will display the data that is generated by the prepopulate adapter automatically.
- If the Auto-prepopulate check box is cleared, an Oracle Identity Manager user must manually trigger the displaying of the data that is generated by the prepopulate adapter. To do this, the administrator must click the prepopulate button on the form section of the direct provisioning wizard in the Web client, while provisioning the form to a user.

Tip: Once you allocate a prepopulate adapter to a form field, and assign a prepopulate rule to the adapter, a quick way to see the association among the adapter, the form field, and the rule is by accessing the Usage Lookup tab of the Adapter Factory form.

3.13.2 Removing Prepopulate Adapters from Form Fields

If a prepopulate adapter, which has been associated with a form field, is no longer valid, you must remove the adapter from the field.

Note: Before removing the prepopulate adapter from a form field, you must create a new version of the form.

To remove a prepopulate adapter from a form field:

1. Select the prepopulate adapter that you want to remove.
2. Click **Delete**. The prepopulate adapter is removed from the form field. It cannot be triggered when the form is launched.
3. After removing the adapter, you must activate the form.

3.14 Working with Process Task Adapters

This section contains these topics:

- Guidelines for Working with a Process Task Adapter
- Attaching Process Task Adapters to Process Tasks
- Removing Process Task Adapters from Process Tasks

3.14.1 Guidelines for Working with a Process Task Adapter

After you create a process task adapter, you attach it to the appropriate process task by using the Integration tab of the Process Definition form. From this tab, you can also map any variables of the adapter to their proper locations, which were designated as either *Resolve at Run time* or as an adapter return variable.

For example, the adapter named *adpSOLARISPASSWORDUPDATED* is connected to the *Password Updated* task of the *Solaris* process.

After you attach an adapter to a process task, for the adapter to be functional, it might need data from fields of other forms. For this example, the *adpSOLARISPASSWORDUPDATED* adapter cannot work unless it obtains the following information:

- The user's Oracle Identity Manager ID and password.
- The user's Solaris ID and password.
- The IP address where Solaris is located.

Therefore, it must get this information from the *UserID*, *Passwd*, *SolarisUserID*, *SolarisUserPasswd*, and *ServerAddress adapter* variables respectively. These five variables are created by using the Adapter Factory form. The "Y" that precedes each adapter variable signifies that it has been mapped correctly.

The form that enables you to create process-specific fields, which will be used by a process to obtain the information it needs, is called the Form Designer. When you create these fields, Oracle Identity Manager stores them into a table. Then, by associating this table with a process (through the Table Name lookup field of the Process Definition form), the adapter, which you attach to a task of this process, will use the table to retrieve the appropriate data.

If you want to modify this table, you can do so through the Form Designer form.

The *UD_SOLARIS* table contains two fields: *UD_SOLARIS_USERID* and *UD_SOLARIS_PASSWD*. By accessing this record of the Form Designer form, you can edit the fields of the table.

Once you attach the process task adapter to a dependent process task, and the status of this process task is *Pending* (the status of the previous process task is *Completed*), Oracle Identity Manager will trigger the adapter automatically. When the process task is an independent task, Oracle Identity Manager will execute the adapter as soon as the process is requested.

The result of the adapter being executed represents the state of the process task. When the adapter is finished successfully, the process task to which this adapter is attached will have a status of *Completed*.

On the other hand, if the adapter cannot perform its designated function, the process task to which this adapter is attached will have a status of *Rejected*. By discovering the cause of the error, you can modify the process task and/or adapter so it can run successfully.

Note: To determine why a process task might have failed:

Find the process task. When the process task has not yet been provisioned to the target user or organization, it is located in the To Do List or Pending Approvals. To find the task:

1. Log in as the user.
2. Select the To Do List link or the Pending Approvals links in the left side of the window.

3.14.2 Attaching Process Task Adapters to Process Tasks

In the previous chapter, you learned how to create a process task adapter. You must attach it to a process task to execute that process task automatically.

To connect an adapter to a process task, access the Integration tab (from the Process Definition form). From this tab, you can also map any adapter variables to their proper locations.

The following procedure shows you how to attach a process task adapter to a process task:

1. Open the Process Definition form, which is located in the Process Management folder.

   In the Oracle Identity Manager Workspace, the Process Definition form appears.

2. Select the process, which contains a task to which you want to attach an adapter. The selected process, along with its tasks, appears in the Process Definition form. For this example, the Solaris process has been selected.

3. Double-click the row header of the task to which you want to attach an adapter. The Editing Task window appears, containing information about the task (for example, the *Password Updated* process task).

4. Click the **Integration** tab.

5. Click **Add**.

   The Handler Selection window appears.

6. To access Oracle Identity Manager adapters, click the **Adapter** option.

   The adapters appear, which you can attach to the process task.

7. From this region, select the adapter that you want to attach to the process task, for example, the adp*SOLARISPASSWORDUPDATED* adapter.

   > **Tip:** For classification purposes, the first three letters of each adapter's name are adp. For classification purposes, the first three letters of each adapter's name are *adp*.

8. From the Handler Selection window's toolbar, click **Save**.

   A dialog box appears, stating that the adapter was successfully added to the process task.

9. Click **OK**.

   The dialog box disappears, and the **Integration** tab is now active. This tab now displays the following:

   - The name of the adapter that is attached to the process task;
   - The status of the adapter; and
   - The names, descriptions, and mapping statuses of the adapter's variables.

   > **Note:** An adapter can have one of three mapping statuses:
   >
   > *Ready*. This adapter has been successfully compiled, and all of its variables have been mapped correctly.
   >
   > *Mapping Incomplete*. This adapter has been successfully compiled, but at least one of its variables have not been mapped correctly.
   >
   > *Adapter Unavailable*. After this adapter had been compiled successfully, it was modified, and recompiled.

Note: If an adapter does not have any mappable variables, the Adapter Variables region is empty. In addition, the Status field will display either *Ready* or *Adapter Unavailable*, depending on whether the adapter has to be recompiled.

Note: A mappable adapter variable either has been designated as *Resolve at Run time* or it is an adapter return variable.

Note: Once you attach the adapter to the process task, any responses that you defined for the adapter appear in the Responses tab of the Editing Task window.

10. Set the mappings for each variable that appears in the Adapter Variables region of the Integration tab. To do so, double-click the row header of the variable you want to map (for example, *SolarisUserID*).

    The Data Mapping for Variable window is displayed.

    Table 3–7 describes the fields of the Data Mapping for Variable window.

Table 3–7 Fields of the Data Mapping for Variable WIndow

Field Name	Description
Variable Name	This field displays the name of the adapter variable for which you are setting a mapping (for example, *SolarisUserID*).
Data Type	This field shows the data type of the adapter variable (for example, *String* is the data type for the *SolarisUserID* variable).
Map To	This field contains the types of mappings that you can set for the adapter variable (for example, *IT Resources*).When you map the adapter variable to a location or a contact, Oracle Identity Manager enables the adjacent combo box. From this combo box, select the specific type of location or contact to which you are mapping the adapter variable. In addition, if you map the adapter variable to a custom process form, and this form contains child table(s), Oracle Identity Manager enables the adjacent combo box. From this combo box, select the child table to which you are mapping the adapter variable. If you are not mapping the adapter variable to a location, contact, or child table of a custom process form, this combo box is grayed out.
Qualifier	This field contains the qualifiers for the mapping you selected in the **Map to** combo box (for example, *IT Asset*).
IT Asset Type	This field enables you to select a specific IT Resource (for example, *Solaris*) when you map an adapter variable to an IT Resource, and this variable's data type is *String*. If you are not mapping the adapter variable to an IT Resource, or the variable's data type is not *String*, this field does not appear.

Table 3–7 (Cont.) Fields of the Data Mapping for Variable WIndow

Field Name	Description
IT Asset Property	This field enables you to select a specific field that will receive the results of the mapping (for example, *User Name*), when you map an adapter variable to an IT Resource, and this variable's data type is *String*. If you are not mapping the adapter variable to an IT Resource, or the variable's data type is not String, this field does not appear. **Important**: The **IT Asset Type** and **IT Asset Property** fields are included within this window for backward compatibility. The preferred way is to create an adapter variable with a data type of *IT Resource,* in which case these fields will not appear.
Literal Value	When you map the adapter variable to a literal, use this field to specify the specific literal value. If you are not mapping the adapter variable to a literal, this field does not appear.
Old Value	By selecting this check box, you map the adapter variable to the value that was originally in the selected Qualifier field before modification. Process task adapters associated with process tasks are conditionally triggered when some field on the process form gets changed. If you click the Old Value option, and the process task is marked Conditional, the value that is passed to the adapter is the previous value of the field, before it got modified. This is useful in cases of fields that accept passwords. For example, if you want to disallow setting the password to the same value, you can use the old value for comparison. If you are not mapping the adapter variable to a field that belongs to a child table of a custom process form, this check box is grayed out.

11. Complete the Map To, Qualifier, IT Asset Type, IT Asset Property, Literal Value, and Old Value fields.

    See Also: "Adapter Mapping Information" on page 3-34 for more information about the mappings to select

12. On the toolbar, click **Save**. Then, click **Close**.

    The Data Mapping for Variable window disappears. The **Integration** tab is active again.

13. On the Editing Task window toolbar, click **Save**.

    The contents in the **Status** field change from *Mapping Incomplete* to *Ready*. In addition, the mapping statuses for the adapter's variables change from *No (N)* to *Yes (Y)*.

14. On the toolbar, click **Close**.

    The Editing Task window disappears, and the main screen is active once again. The adapter you added to the *Password Updated* task (*adpSOLARISPASSWORDUPDATED*) appears in the **Process Definition** form.

    This signifies that the *adpSOLARISPASSWORDUPDATED* process task adapter was attached to the *Password Updated* process task.

Tip: Once you attach a process task adapter to a process task, a quick way to see the process and task to which it is connected is by accessing the Usage Lookup tab of the Adapter Factory form.

3.14.3 Removing Process Task Adapters from Process Tasks

If a process task adapter is no longer necessary for Oracle Identity Manager to complete the process task automatically, or when you wish to attach a different adapter to a process task, you must first remove the adapter that is attached to the process task.

This procedure will show you how to remove a process task adapter from a process task.

3.14.3.1 To Remove a Process Task Adapter from a Process Task

1. Open the Process Definition form.

   In the Design Console workspace, the **Process Definition** form appears.

2. Select the process, which contains a task from which you want to remove an adapter (for example, the *Solaris* process).

   The selected process, along with its tasks, appears in the **Process Definition** form.

3. Double-click the row header of the process task from which you want to remove the adapter (for example, the *Password Updated* task).

   The Editing Task window appears, containing information about the process task. Click the Integration tab.

4. Click the **Integration** tab.

   The Integration tab displays information about the adapter that is attached to the process task.

5. Click **Remove**.

   A dialog box appears, asking if you want to remove the adapter from the process task.

6. Click **OK**.

   A dialog box appears, signifying that the adapter has been removed from the process task.

7. Click **OK**.

   The contents of the adapter no longer appear in the Integration tab.

8. On the toolbar, click **Close**.

   The Editing Task window disappears, and the main screen is active once again. The adapter that was once linked to the *Password Updated* task *(adpSOLARISPASSWORDUPDATED)* no longer appears in the child table of the **Process Definition** form.

   This signifies that you have removed the adapter from the process task.

3.15 Adapter Mapping Information

An adapter is a Java class, generated by the Adapter Factory, which enables Oracle Identity Manager to interact with an external JAR file, a target IT resource (for example, a resource asset), or a user-defined form. The Adapter Factory is a code-generation tool provided by Oracle Identity Manager, which enables a User Administrator to create Java classes.

An adapter extends the internal logic and functionality of Oracle Identity Manager. It automates process tasks, and defines the rules for the auto-generation and validation of data in fields within Oracle Identity Manager. There are five types of adapters: task assignment adapters, task adapters, rule generator adapters, pre-populate adapters, and entity adapters.

The following topics are discussed in this section:

- Adapter Task Mapping Information
- Adapter Variable Mapping Information

3.15.1 Adapter Task Mapping Information

An adapter task is one of the several possible components within an adapter. And this is a logical step within an adapter, equivalent to calling a programming language method. The following types of adapter tasks are available: Functional Tasks (Java Task), Utility Tasks (Utility Task and Oracle Identity Manager API Task), and Logic Tasks (Set Variable Task and Error Handler Task).

This section lists the mappings that you can set for the parameters of an adapter task, in the following topics:

- Adapter Variables
- Adapter Task
- Literal
- Adapter References
- Process Definition
- User Definition

3.15.1.1 Adapter Variables

The following table lists and describes the items of the Map To list box of the Data Mapping for Variable window and the Name list box to which you can map the parameters of an adapter variable for an adapter task.

Map To Combo Box	Name Combo Box	Description
Adapter Variables	A list of adapter variables are displayed	You can map the parameter to the adapter variables that you created for this adapter. **Note**: When the adapter variable's classification type is Object, it cannot be used with process task adapters. **Note**: If the adapter variable's classification type is IT Resource, then an Attribute combo box is displayed. From this combo box, select the attribute of the IT resource to which you wish to map the parameter.

3.15.1.2 Adapter Task

The following table lists and describes the items of the Map To, Name, and Output combo boxes of the Adapter Factory form to which you can map the parameters of an adapter task.

Map To Combo Box	Name Combo Box	Output combo Box	Description
Adapter Task	A list of adapter tasks are displayed.	A list of output variables pertaining to the selected adapter task is displayed.	You can map the parameter to the adapter tasks that you created for this adapter.

3.15.1.3 Literal

The following table lists and describes the items of the Map To and Type combo boxes, as well as the Value field of the Adapter Factory form, to which you can map the parameters of a constant (or literal) for an adapter task.

Map To Combo Box	Type Combo Box	Value Field	Description
Literal	String, Boolean, Character, Byte, Date, Integer, Float, Long, Short, Double	Enter the value of the literal into this field.	You can map the parameter to a String, Boolean, Character, Byte, Date, Integer, Float, Long, Short, or Double data type, respectively.

3.15.1.4 Adapter References

The following table lists and describes the items of the Map To and Type combo boxes of the Adapter Factory form to which you can map the parameters of an adapter reference for an adapter task.

Map To Combo Box	Type Combo Box	Description
Adapter References	Event Handler Name or Database Reference	You can map the parameter to the active adapter.

3.15.1.5 Process Definition

The following table lists and describes the items of the Map To and Field combo boxes of the Adapter Factory form to which you can map the parameters of a process definition for an adapter task.

Map To Combo Box	Field Combo Box	Description
Process Definition	Name	You can map the parameter to the Name field of the Process Definition form.
	Type	You can map the parameter to the Type field of the Process Definition form.

3.15.1.6 User Definition

The following table lists and describes the items of the Map To and Field combo boxes of the Adapter Factory form to which you can map the parameters of a user definition for an adapter task.

Map To Combo Box	Field Combo Box	Description
User Definition	User Key	You can map the parameter to a key, representing a unique record of the Users form.
	First Name	You can map the parameter to the First Name field of the Users form.
	Middle Initial	You can map the parameter to the Middle Name field of the Users form.
	Last Name	You can map the parameter to the Last Name field of the Users form.
	User Login	You can map the parameter to the User ID field of the Users form.
	Password	You can map the parameter to user password of the Users form.
	Type	You can map the parameter to the Xellerate Type field of the Users form.
	User Status	You can map the parameter to the Status field of the Users form.
	Role	You can map the parameter to the Role field of the Users form.
	Identity	You can map the parameter to the Identity field of the Users form.
	Disabled	You can map the parameter to the Disable User check box of the Users form.
	Organization	You can map the parameter to the Organization field of the Users form.
	Manager	You can map the parameter to the Manager field of the Users form.
	Start Date	You can map the parameter to the Start Date field of the Users form.
	End Date	You can map the parameter to the End Date field of the Users form.

Map To Combo Box	Field Combo Box	Description
	Email	You can map the parameter to the Email field of the Users form.
	Provisioning Date	You can map the parameter to the Provisioning Date field of the Users form.
	Provisioned Date	You can map the parameter to the Provisioned Date field of the Users form.
	Deprovisioning Date	You can map the parameter to the Deprovisioning Date field of the Users form.
	Deprovisioned Date	You can map the parameter to the Deprovisioned Date field of the Users form.
	Any fields that are displayed in the User Defined Fields tab of the Users form.	You can map the parameter to the selected user-defined field.

3.15.2 Adapter Variable Mapping Information

For a newly created adapter to work, you can map data to the parameters of the adapter's tasks. For this reason, you create placeholders, also known as adapter variables, to map the data at run time. Once an adapter variable is not needed for the adapter to run, you can remove it from the adapter. After you have deleted the adapter variable, recompile the adapter.

When an adapter variable is not the adapter return variable, or it is not designated as Resolve at Run time, it should be mapped within the Variable List tab of the Adapter Factory form. On the other hand, if the adapter variable is classified as an adapter return variable, or the adapter variable is set to Resolve at Run time, it can be mapped at another location within Oracle Identity Manager. This location is contingent upon the adapter's type. For example, the variables of a process task adapter will be mapped at a different place than the variables of a pre-populate adapter. The following table lists the variables of a particular type of adapter that can be mapped.

Adapter Type	Location
Process Task	The Integration tab of the Editing Task window
Task Assignment	The Assignment tab of the Editing Task window
Rule Generator	The Map Adapters tab of the Data Object Manager form
Pre-Populate	The Pre-Populate tab of the Form Designer form
Entity	The Map Adapters tab of the Data Object Manager form

The following topics are discussed in this section:

- From the Variable List Tab
- Process Task Adapter Variable Mappings
- Task Assignment Adapter Variable Mappings
- Rule Generator and Entity Adapter Variable Mappings
- Prepopulate Adapter Variable Mappings

3.15.2.1 From the Variable List Tab

The following table lists the mappings that you can set from the Variable List tab.

Variable Type	Map To	Qualifier/Resource Type
Object	Adapter References	Database References
		Data Object References
	Set at run time (for Task Assignment adapters only)	Database References
		Data Object References
IT Resource	Resolve at Run time	The IT Resource types that are displayed in the Table view of the IT Resources Type Definition form
String, Character, Byte, Integer, Float, Long, Short, Double	Literal	If you are mapping the adapter variable to a literal, a Literal Value field is displayed below the Resource Type combo box. Within this field, enter the value of this literal.
	Resolve at Run time	NA
	Adapter References	Event Handler Name **Note**: If the data type of the adapter variable is not String, Adapter References cannot be selected from the Map To combo box.
Boolean	Literal	Boolean. If you select this resource type, two Literal Value options are displayed below the Resource Type combo box: True and False. Select the option that corresponds to the value of the adapter variable.
	Resolve at Run time	NA
Date	Literal	If you are mapping the adapter variable to a literal, a Literal Value lookup field is displayed below the Resource Type combo box. Double-click this lookup field. From the Date & Time window that is displayed, select the date and time that will be the value of this literal.
	Resolve at Run time	NA
	System Date	NA **Note**: This variable's value will reflect Oracle Identity Manager's date and time. Hence, you do not map it.

3.15.2.2 Process Task Adapter Variable Mappings

The following table lists the process task adapter variable mappings.

Variable Type	Map To	Qualifier/Description
Object (Adapter Return Variable)	Process Data	You can map the parameter to a field of either the associated custom process form, or a child table that belongs to this form.
	Response Code	NA
	Task Information	**Note**. You can map the parameter to the Note tab of the Task List form. **Reason**. You can map the parameter to the Error Details window. To access this window, double-click a task that is displayed within the Task List form.
	Process Definition	**Name**. You can map the parameter to the Name field of the Process Definition form. **Type**. You can map the parameter to the Type lookup field of the Process Definition form.
Object (Adapter Return Variable)	Organization Definition	The fields of the Organizations form to which you can map the adapter variable. **Note**: Because the data type of the adapter variable is Object, you cannot select Organization ID from the Qualifier combo box.
	User Definition	The fields of the Users form to which you can map the adapter variable.
IT Resource	IT Resource	You can map the parameter to an IT resource. This IT resource is a member of the IT resource type that is displayed in parenthesis from within the Data Type field.
	Process Data	You can map the parameter to a field of the associated process-specific form. **Note**: The only field names that are displayed in this combo box are ones with a data type of IT Resource Lookup Field.
String, Boolean, Character, Byte, Date, Integer, Float, Long, Short, Double	Process Data	You can map the parameter to a field of either the associated custom process form, or a child table that belongs to this form.
	Task Information	**Note**. You can map the parameter to the Note tab of the Task List form. **Reason**. You can map the parameter to the Error Details window. To access this window, double-click a task that is displayed within the Task List form.
	Process Definition	**Name**. You can map the parameter to the Name field of the Process Definition form. **Type**. You can map the parameter to the Type lookup field of the Process Definition form.

Variable Type	Map To	Qualifier/Description
	Organization Definition	The fields of the Organizations form to which you can map the adapter variable.
String, Boolean, Character, Byte, Date, Integer, Float, Long, Short, Double	User Definition	The fields of the **Users** form to which you can map the adapter variable.
	Literal	If you are mapping the adapter variable to a literal, and the variable's data type is String, Character, Byte, Integer, Float, Long, Short, or Double, a Literal Value field is displayed below the Qualifier combo box. Within the field, enter the value of this literal. When you are mapping the adapter variable to a literal, and the variable's data type is Boolean, two Literal Value options are displayed below the Qualifier combo box: True and False. Select the option that corresponds to the value of the adapter variable. If you are mapping the adapter variable to a literal, and the variable's data type is Date, a Literal Value lookup field is displayed below the Qualifier combo box. Double-click this lookup field. From the Date & Time window that is displayed, select the date and time that will be the value of this literal.
String	IT Resources	If you are mapping the adapter variable to an IT Resource, three combo boxes are displayed below the Map To combo box: Qualifier, IT Asset Type, and IT Asset Property. From these combo boxes, select the qualifier for the mapping, the specific name of the IT resource, and the field of the IT resource that will receive the results of the mapping. Note: If the data type of the adapter variable is not String, IT Resources cannot be selected from the Map To combo box.

3.15.2.3 Task Assignment Adapter Variable Mappings

The following table lists the task assignment adapter variable mappings.

Variable Type	Map To	Qualifier/Description
IT Resource	Object Data	You can map the parameter to an IT resource's instance key. This IT resource is a member of the IT resource type that is displayed in parenthesis from within the Data Type field.
	IT Resource	You can map the parameter to an IT resource.

Variable Type	Map To	Qualifier/Description
Object (Adapter Return Value)	Object Data	You can map the parameter to a field of either the associated custom resource object form, or a child table that belongs to this form.
	Response Code	NA
	Task Information	The fields of the Task List form to which you can map the adapter variable.
	Process Definition	The fields of the Process Definition form to which you can map the adapter variable.
	Organization Definition	The fields of the Organizations form to which you can map the adapter variable.
	User Definition	The fields of the Users form to which you can map the adapter variable.
String, Boolean, Character, Byte, Date, Integer, Float, Long, Short, Double	Object Data	You can map the parameter to a resource object's instance key.
	Task Information	The fields of the Task List form to which you can map the adapter variable.
	Process Definition	The fields of the Process Definition form to which you can map the adapter variable.
	Organization Definition	The fields of the Organizations form to which you can map the adapter variable.
String, Boolean, Character, Byte, Date, Integer, Float, Long, Short, Double	User Definition	The fields of the Users form to which you can map the adapter variable.
	Request Info	Request ID. You can map the parameter to the Request ID field of the Requests form.
		Request Action. You can map the parameter to the Request Action field of the Requests form.
		Request Priority. You can map the parameter to the Request Priority field of the Requests form.
	Request Target User	The fields of the Users form to which you can map the adapter variable.
	Request Target Organization	The fields of the Organizations form to which you can map the adapter variable.
	Requester Info	The fields of the Users form to which you can map the adapter variable.

Variable Type	Map To	Qualifier/Description
	Literal	If you are mapping the adapter variable to a literal, a Literal Value field is displayed below the Qualifier combo box. Within the field, enter the value of this literal. **Note**: If the data type of the adapter variable is Boolean, two options are displayed in place of the field: True and False. Select the option that reflects the value of the adapter variable. **Note**: If the data type of the adapter variable is Object, Literal cannot be selected from the Map To combo box.
String	IT Resources	Resource Instance. You can map the parameter to an IT resource's instance key. This IT resource is a member of the IT resource type that is displayed in parenthesis from within the Data Type field. IT Asset Type. You can map the parameter to an IT resource type.
String	IT Resources	IT Asset Property. You can map this parameter to one of the properties that comprise the selected IT resource type.

3.15.2.4 Rule Generator and Entity Adapter Variable Mappings

The following table lists the rule generator and entity adapter variable mappings.

Variable Type	Map To	Qualifier/Description
Object (Adapter Return Variable), IT Resource, String, Boolean, Character, Byte, Date, Integer, Float, Long, Short	Literal	If you are mapping the adapter variable to a literal, a Literal Value field is displayed below the Qualifier combo box. Within the field, enter the value of this literal. **Note**: If the data type of the adapter variable is Object, Literal cannot be selected from the Map To combo box.
	Entity Field	You can map the adapter variable to a field of the associated process form. The name of this form is displayed in the Form Description field of the Data Object Manager form.
	Organization Definition	The fields of the Organizations form to which you can map the adapter variable. **Note**: If the data type of the adapter variable is not Object, you cannot select Organization ID and Organization Parent ID from the Qualifier combo box.
	User Definition	The fields of the Users form to which you can map the adapter variable.

3.15.2.5 Prepopulate Adapter Variable Mappings

The following table lists the prepopulate adapter variable mappings.

Variable Type	Map To	Qualifier/Description
IT Resource	IT Resource	You can map the parameter to an IT resource. This IT resource is a member of the IT resource type that is displayed in parenthesis from within the Data Type field.
	Process Data	You can map the parameter to a field of the associated process-specific form. **Note**: The only field names that are displayed in this combo box are ones with a data type of IT Resource Lookup Field.
Object, String, Boolean, Character, Byte, Date, Integer, Float, Long, Short, Double	Process Data	You can map the parameter to a field of the associated process-specific form.
	Organization Definition	The fields of the Organizations form to which you can map the adapter variable.
	User Definition	The fields of the Users form to which you can map the adapter variable.
String, Boolean, Character, Byte, Date, Integer, Float, Long, Short, Double	Literal	If you are mapping the adapter variable to a literal, and the variable's data type is String, Character, Byte, Integer, Float, Long, Short, or Double, a Literal Value field is displayed below the Qualifier combo box. Within the field, enter the value of this literal. When you are mapping the adapter variable to a literal, and the variable's data type is Boolean, two Literal Value options are displayed below the Qualifier combo box: True and False. Select the option that corresponds to the value of the adapter variable. If you are mapping the adapter variable to a literal, and the variable's data type is Date, a Literal Value lookup field is displayed below the Qualifier combo box. Double-click this lookup field. From the Date & Time window that is displayed, select the date and time that will be the value of this literal.

Variable Type	Map To	Qualifier/Description
String	IT Resources	If you are mapping the adapter variable to an IT Resource, three combo boxes are displayed below the Map To combo box: Qualifier, IT Asset Type, and IT Asset Property. From these combo boxes, select the qualifier for the mapping, the specific name of the IT resource, and the field of the IT resource that will receive the results of the mapping. **Note**: If the data type of the adapter variable is not String, then IT Resources cannot be selected from the Map To combo box.

3.16 Defining Error Messages

The Error Message Definition form, as shown in Figure 3–3, is in the Development Tools folder of the Design Console. It is used to:

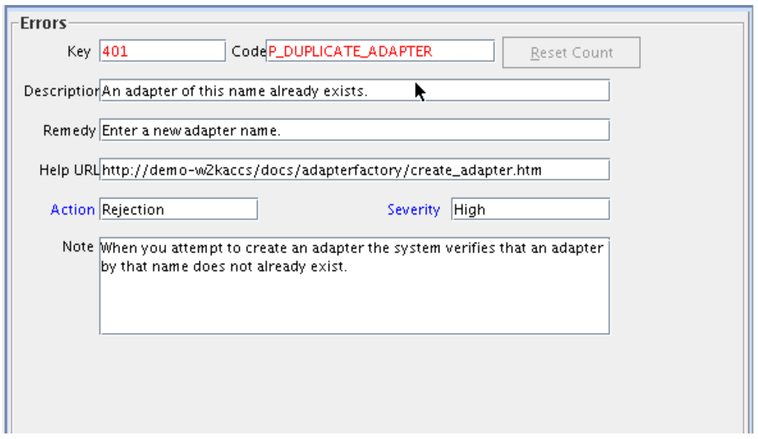
- Create the error messages that are displayed in dialog boxes when certain problems occur.
- Define the error messages that users can access when they create error handler tasks by using the Adapter Factory form.

  The error messages you create are displayed on the Identity Self Service or Identity System Administration if they are added to an adapter definition while creating a new adapter by using an error handler logic task based on a failure condition.

Note: If an entity adapter is attached to a process form or an object form for validation of field values, these adapters will run if you edit data in these forms after completing direct or request provisioning.

Oracle Identity Manager 11*g* Release 2 (11.1.2.3.0) does not support creating new entity adapters.

Figure 3–3 Error Message Definition Form



Key	401
Code	P_DUPLICATE_ADAPTER
Description	An adapter of this name already exists.
Remedy	Enter a new adapter name.
Help URL	http://demo-w2kaccs/docs/adapterfactory/create_adapter.htm
Action	Rejection
Severity	High
Note	When you attempt to create an adapter the system verifies that an adapter by that name does not already exist.

Table 3–8 describes the data fields of the Error Message Definition form.

Table 3–8 Fields of the Error Message Definition Form

Field Name	Description
Key	The error message definition's unique, system-generated identification number.
Code	The code that represents the error message definition.
Reset Count	When you click this button, Oracle Identity Manager resets the counter to zero. This counter is the number of times the error message is displayed.
Description	A description of the error message.
Remedy	A description of how to correct the condition that caused the error message to be displayed.
Help URL	The link to the URL that contains an online Help topic for this error message.
Action	A one-letter code, representing the seriousness of the condition that causes the error message to be displayed. An error message has three levels of seriousness: Error (E), Rejection (R), and Fatal Rejection (F).
Severity	For classification purposes, you can categorize the seriousness of the condition that results in the error message being displayed, even further. An error message has five sub-levels of severity: None (N), Low (L), Medium (M), High (H), and Crash (C).
Note	Explanatory information about the error message.

When you create an error message, Oracle Identity Manager populates the **Key** field with a unique identification number. When a condition occurs that causes the error message to be displayed, the text in the **Description** field is displayed in a dialog box.

Note: After you create an error message definition, to reset the count of how many times the error message is displayed, click the **Reset Count** button. This resets the count to zero.

To create an error message:

1. Open the Error Messaging Definition form.
2. In the **Code** field, enter the code that represents the error message definition.
3. In the **Description** field, enter a description for the error message.
4. In the **Remedy** field, you can enter a description for how to correct the condition that causes the error message to be displayed.
5. In the **Help URL** field, you can enter the link to the URL that contains an online Help topic for this error message.
6. (Optional) Double-click the **Action Lookup** field.

   From the Lookup dialog box that is displayed, you can select a code that represents the seriousness of the condition that causes the error message to be displayed. These codes, listed by degree of seriousness (from lowest to highest), are:

- Error (E). Oracle Identity Manager stores the error message, and stops any related operations from being triggered. Instead, the operation rolls back to the previous operation.
- Reject (R). Oracle Identity Manager stores the rejection message, but it does not prevent subsequent operations from being executed.
- Fatal Reject (F). Oracle Identity Manager stores the rejection message, and it stops any subsequent operations from being triggered. However, it stores all operations that were executed up to the fatal rejection.

7. (Optional) Double-click the **Severity Lookup** field. From the Lookup dialog box that is displayed, you can select a code (None (N), Low (L), Medium (M), High (H), or Crash (C)). This code presents a detailed classification of the code that is displayed in the **Action** lookup field.
8. In the **Note** field, enter explanatory information about the error message.
9. Click **Save**.

   The error message is created.

   After creating error messages by using the Error Message Definition form, you must add new error codes and advice messages in the Oracle Identity Manager `customResources.properties` resource bundle. These localized error codes and advice messages will be displayed in Identity Self Service or Identity System Administration.

4 Understanding the Identity Connector Framework

Identity connectors are components developed to link Oracle Identity Manager with external stores of applications, directories, and databases. Oracle Identity Manager provides support for developing and building identity connectors by using the Identity Connector Framework (ICF). ICF decouples Oracle Identity Manager from other applications to which it connects. Therefore, you can build and test an identity connector before integrating it with Oracle Identity Manager. This chapter contains conceptual information and sample code in the following sections:

Note: Earlier releases of Oracle Identity Manager have other options for building identity connectors. These options are still supported, but it is recommended that you build new identity connectors by using the ICF.

4.1 Advantages of ICF

ICF provides the following benefits:

- Single platform: Identity Connectors are shared between Oracle Identity Manager and Oracle Waveset (OW), which means they are built on top of the same platform so that a single connector can be used for both Oracle Identity Manager and OW to communicate with external identity-aware applications.
- Simple installation: ICF offers simple installation as most of the manual configuration during installation, such as copying the connector files and the external code files are automatically taken care by ICF.
- Stateless by design: Identity connectors are stateless by design. An identity connector stores nothing. The calling application supplies to the connector the values for its configuration, including the information required to connect to the target application. This is because, identity connectors are stateless, each bundle

implementation are kept as simple as possible, and coupling the implementation with that of the calling application is also prevented.

- ICF Common: ICF provides common connector integration layer for all ICF based connectors in Oracle Identity Manager and no development effort is required to develop ICF Common.
- Remote Execution: ICF supports remote execution of connector server using Java or .NET implementation.
- JVM Isolation: Remote ICF provides JVM isolation, which means running a Java connector on a different host avoids JVM conflicts.
- Reuse: In future, other products can reuse Identity Connectors.

4.2 Introducing the ICF Architecture

Identity connectors allow Oracle Identity Manager to carry out user provisioning and reconciliation operations on target systems in the enterprise. ICF decouples any calling application, such as Oracle Identity Manager, from the implementation of the connector. ICF also decouples the implementation of the connector from the calling application. The same connector implementation can work with several different calling applications. Figure 4–1 illustrates how this is accomplished by situating the ICF API and SPI between Oracle Identity Manager and the target system.

The API implementation always post-processes the results returned by the SPI Search operation. This double-checks the SPI implementation if the connector bundle does not implement all Filter types, or does not implement them properly for all attributes. If the implementation of Search in the SPI returns every object of the specified type, then the API implementation discards every object that does not match the specified Filter. Post-processing in the API implementation is expensive in terms of processing-time and network-bandwidth, and therefore, it is more efficient if each connector-bundle supports every type of filter (search predicate or logical operator) that the target application can support natively. See the details for Filter Translator in "Common Classes" on page 4-21.

Figure 4–1 illustrates that the calling application sees only the ICF API. The ICF API dedicates a classloader to each connector bundle, so that the calling application is not exposed to the classes and libraries in the implementation of the connector-bundle (SPI). Bundle classloader also ensures isolation between the bundles as well as making any bundled library available to the connector bundle only, thereby avoiding conflicts between dependencies.

Figure 4–1 Identity Connector Framework Deployment

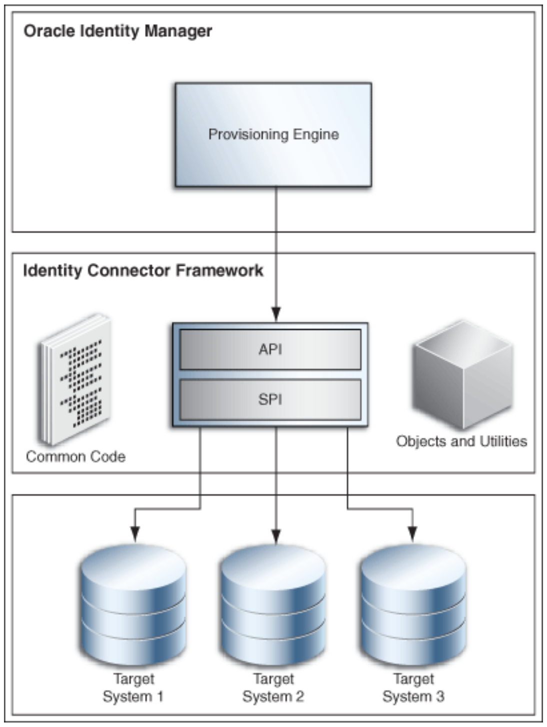


Figure 4–2 illustrates the backwards compatibility of the ICF. Newer bundles may be deployed without affecting existing ones. In addition, newer versions of the ICF are generally backward-compatible with existing bundles. Therefore, any connector should work with a new version of framework.

Figure 4–2 Compatibility Between the ICF and Connector Bundles

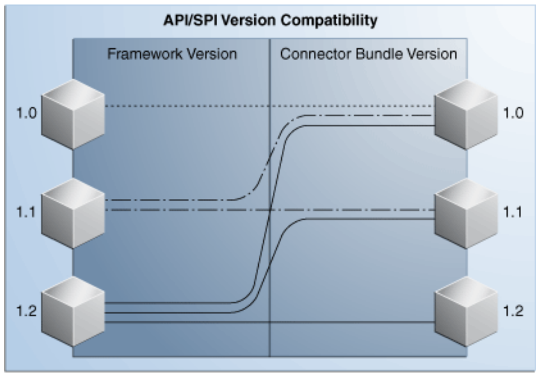


Figure 4–3 illustrates deployment methodology of the ICF. Framework supports LCM to clone connector to support multiple versions of the same target. In addition, Framework supports connection pooling.

Figure 4–3 Deployment Methodology to Support Multiple Versions of Same Target

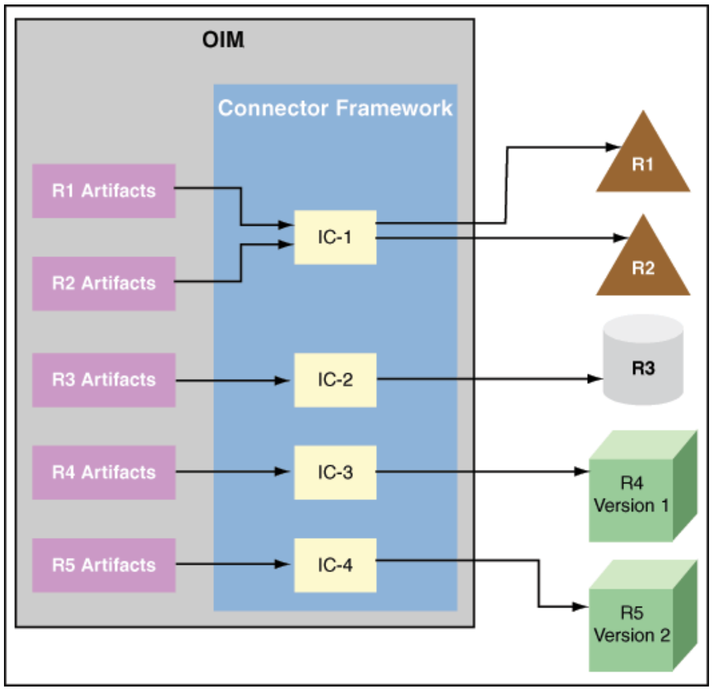


Figure 4–4 illustrates Framework installed on remote system. This enables remote execution of connector server using Java or .NET implementation with targets being local or remote to connector bundles. This is required when a connector bundle is not directly executed with in an application and ICF allows the application to communicate with externally deployed bundles. In addition, the connector artifacts can be same for local or remote system.

Figure 4–4 Connector Server Remote System Framework

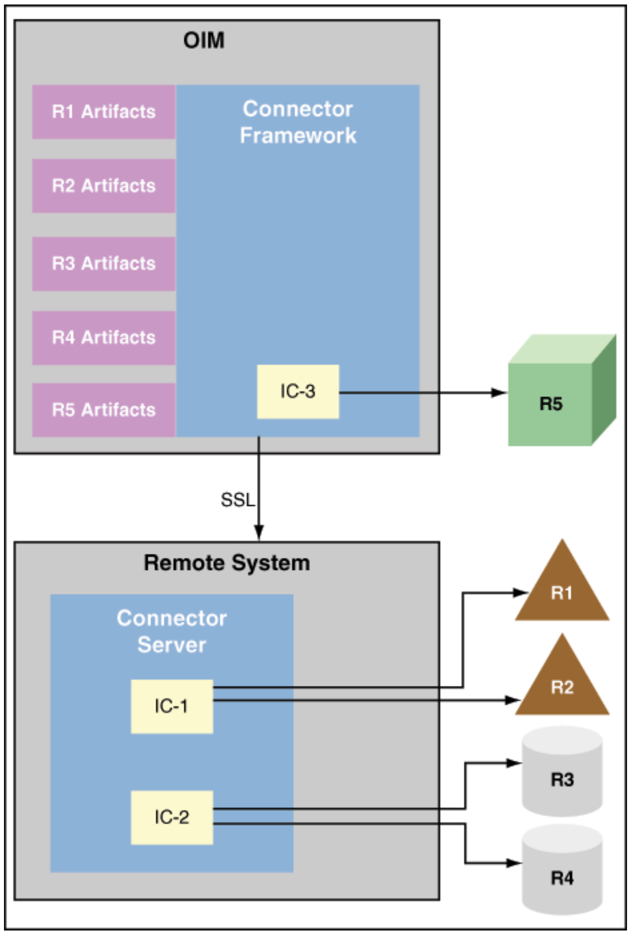
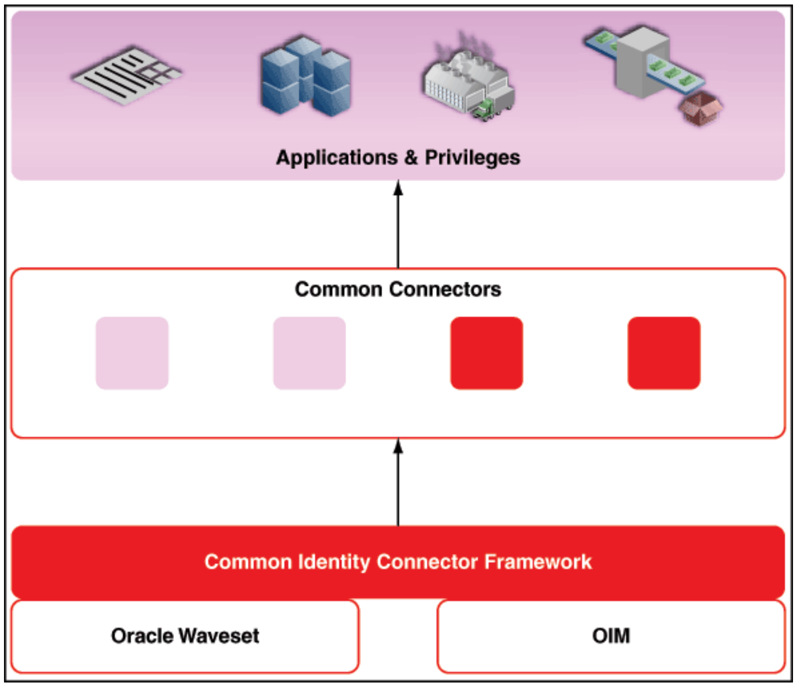


Figure 4–5 illustrates ICF Framework, which enables the convergence of Oracle Identity Manager and Oracle Waveset (OW) connectors to a single connector, best of both.

Figure 4–5 ICF Framework


Applications & Privileges
Common Connectors
Common Identity Connector Framework
Oracle Waveset
OIM

4.3 Using the ICF API

The org.identityconnectors.framework.api package contains the ICF API. Oracle Identity Manager uses the API to call Connector implementations. The API provides a consistent view of any implemented Connector, regardless of the supported operations. The following sections explain these interfaces and classes.

- The ConnectorInfoManagerFactory Class
- The ConnectorInfoManager Interface
- The ConnectorKey Class
- The ConnectorInfo Interface
- The APIConfiguration Interface
- The ConfigurationProperties Interface
- The ConnectorFacadeFactory Class
- The ConnectorFacade Interface

4.3.1 The ConnectorInfoManagerFactory Class

The ConnectorInfoManagerFactory class allows Oracle Identity Manager to load Connector classes from a set of bundles. The static getInstance method returns an object of type ConnectorInfoManagerFactory. This object can then be used to get a reference to the ConnectorInfoManager. (See Section 4.3.2, "The ConnectorInfoManager Interface" for more information.) Example 4–1 illustrates the ConnectorInfoManagerFactory implementation.

Example 4–1 ConnectorInfoManagerFactory Implementation

```
//create ConnectorInfoManagerFactory
```

```
ConnectorInfoManagerFactory cInfoManagerFactory =
    ConnectorInfoManagerFactory.getInstance();
```

4.3.2 The ConnectorInfoManager Interface

The ConnectorInfoManager interface maintains a list of ConnectorInfo instances. Each instance describes an identity connector. ConnectorInfoManager can be obtained by calling the getLocalManager method on the ConnectorInfoManagerFactory, and a list of bundle URLs is passed to it. ConnectorInfoManager can also by obtained by calling getRemoteManager method on the ConnectorInfoManagerFactory. The getRemoteManager method accepts an instance of RemoteFrameworkConnectionInfoand, which is used for getting information about connectors deployed on Connector Server.

In Example 4–2, cInfoManagerFactory is the instance of the ConnectorInfoManagerFactory and bundleURL is a list of bundle URLs that may point to directories consisting of JAR-ed or un-JAR-ed bundles.

Example 4–2 ConnectorInfoManager Implementation

```
//get the ConnectorInfoManager
ConnectorInfoManager cInfoManager =
    cInfoManagerFactory.getLocalManager(bundleURL);
```

4.3.3 The ConnectorKey Class

A ConnectorKey uniquely identifies a Connector instance within an installation. The ConnectorKey class takes a bundleName (name of the Connector bundle), a bundleVersion (version of the Connector bundle) and a connectorName (name of the Connector bundle) as illustrated in Example 4–3.

Example 4–3 ConnectorKey Implementation

```
//get the ConnectorKey reference
ConnectorKey flatFileConnectorKey =
    new ConnectorKey(bundleName, bundleVersion, connectorName);
```

4.3.4 The ConnectorInfo Interface

The ConnectorInfo interface contains information about a specific identity connector. It contains the display name, key and message details regarding the particular identity connector. Example 4–4 illustrates how to implement the ConnectorInfo.

Example 4–4 ConnectorInfo Implementation

```
//get the ConnectorInfo
ConnectorInfo info =
    cInfoManager.findConnectorInfo(flatFileConnectorKey);
```

In the example, cInfoManager is the ConnectorInfoManager and flatFileConnectorKey is the identity connector key.

4.3.5 The APIConfiguration Interface

The APIConfiguration interface shows the configuration properties from both the SPI and the API sides. The getConfigurationProperties method returns a ConfigurationProperties instance based on the connector Configuration implementation, initialized to the defaults. Caller can then modify the properties, as required. Example 4–5 illustrates this.

Example 4–5 APIConfiguration Definition

```
APIConfiguration apiConfig =
    info.createDefaultAPIConfiguration();
```

4.3.6 The ConfigurationProperties Interface

The ConfigurationProperties interface encapsulates the SPI Configuration and uses reflection to identify the individual properties that are available for an application to manipulate. Set all of the identity connector's configuration properties using the setPropertyValue method as defined in Example 4–6.

Example 4–6 setPropertyValue Method Signature

```
public void setPropertyValue
  (java.lang.String name, java.lang.Object value)
```

Example 4–7 illustrates an implementation of the ConfigurationProperties interface.

Example 4–7 ConfigurationProperties Implementation

```
//get the default APIConfiguration
ConfigurationProperties flatFileConfigProps =
    apiConfig.getConfigurationProperties();
```

4.3.7 The ConnectorFacadeFactory Class

The ConnectorFacadeFactory class allows an application to get a Connector instance and to manage a pool of Connector instances. Example 4–8 illustrates the ConnectorFacadeFactory definition.

Example 4–8 ConnectorFacadeFactory Definition

```
//get a reference to ConnectorFacadeFactory
ConnectorFacadeFactory facadeFactory =
    ConnectorFacadeFactory.getinstance();
```

4.3.8 The ConnectorFacade Interface

The ConnectorFacade interface is used by the target system to invoke identity connector operations by representing a specific identity connector on the API side. Example 4–9 illustrates the ConnectorFacade implementation.

Example 4–9 ConnectorFacade Implementation

```
//create a ConnectorFacade (nothing but a reference to Connector on SPI side)
ConnectorFacade connectorFacade = facadeFactory.newInstance(apiConfig)
```

4.4 Introducing the ICF SPI

Developers implement the ICF SPI to create identity connectors. The ICF SPI is made up of many interfaces but the developer need only implement those supported by the target system. SPI can again be classified into required, operation, and feature-based interfaces. Required interfaces must be implemented irrespective of the operations supported and they help to create the connector and maintain the connection with the target system, while operation interfaces help the connector to support various operations. Feature-based interfaces support certain features supported by the ICF.

The following sections have more information.

- Implementing the Required Interfaces
- Implementing the Feature-based Interfaces
- Implementing the Operation Interfaces
- Common Classes

4.4.1 Implementing the Required Interfaces

All identity connectors are required to provide an implementation of two interfaces. These two interfaces declare and initialize the identity connector with the target system. The following sections have more information.

- org.identityconnectors.framework.spi.Connector
- org.identityconnectors.framework.spi.Configuration

4.4.1.1 org.identityconnectors.framework.spi.Connector

This is the main interface to declare an identity connector. Many connectors create the connection to the target system when the connection is required, removing the connection when finished with it, and disposing of any resources it has used. The interface provides the init and dispose life cycle methods for this purpose.

Note: Connector implementations must be annotated with type org.identityconnectors.framework.spi.ConnectorClass by providing the configurationClass and displayNameKey information. The displayNameKey must be a key defined in the Messages.properties file.

Every connector implementation must be annotated with @ConnectorClass. This is required because the ICF would scan all top level .class files in the connector bundle looking for classes that have the @ConnectorClass annotation, therefore, autodiscovering connectors that are defined in the bundle. This annotation requires the following elements:

- **configurationClass:** This is the configuration class for this connector. This class has all the information about the target that can be used by the connector to connect and perform various provisioning and reconciliation operations. See section "org.identityconnectors.framework.spi.Configuration" on page 4-12 for more information on how to implement the configuration class.
- **displayNameKey:** Display name key that must be present in the message catalog.

Example 4–10 is a sample connector implementation.

Example 4–10 Flat File Connector Implementation

```
/**
 * Flat file connector implementation. This connector supports create,
 * delete, search and update operations.
 */
@ConnectorClass
  (configurationClass=FlatFileConfigurationImpl.class,
   displayNameKey="FLAT_FILE_CONNECTOR")
public class FlatFileConnector implements Connector,
   CreateOp, DeleteOp,SearchOp<Map<String, String>>,UpdateOp{
```

In Example 4–10:

- **CreateOp:** Helps the connector to create an entity on the target system
- **DeleteOp:** Helps the connector to delete an entity on the target system
- **SearchOp:** Helps the connector to search an entity on the target system
- **UpdateOp:** Helps the connector to update an existing entity on the target system

See "Implementing the Operation Interfaces" on page 4-15 for more information.

The following sections contain information and sample code that illustrates how you might implement the Connector methods. For complete code regarding a Connector implementation, see "Developing a Flat File Connector" on page 5-1.

- Implementing the init Method
- Implementing the dispose Method
- Implementing the getConfiguration Method

4.4.1.1.1 Implementing the init Method

The init method initializes the connector. The connector initializes itself with the configuration instance as provided with the annotation @ConnectorClass. The init method takes a Configuration object as an argument. The Configuration object has all the information required by the Connector to connect to the target system.

Example 4–11 illustrates how to implement the init method of interfaces in JDK 1.6.

Note: In this document, all code samples use the methods implementing interfaces in JDK 1.6.

Example 4–11 init Method Implementation

```
@Override
 public void init(Configuration config) {
     this.flatFileConfig = (FlatFileConfiguration) config;

     FlatFileIOFactory flatFileIOFactory =
       FlatFileIOFactory.getInstance(flatFileConfig);
     this.flatFileMetadata = flatFileIOFactory.getMetadataInstance();
     this.flatFileParser = flatFileIOFactory.getFileParserInstance();
     this.flatFileWriter = flatFileIOFactory.getFileWriterInstance();
     log.ok("Initialization done");
 }
```

Note: FlatFileIOFactory, FlatFileMetadata, FlatFileParser and FlatFileWriter are supporting classes and are not part of the ICF. An implementation of these classes is illustrated in "Developing a Flat File Connector" on page 5-1.

The init method implementation shown in Example 4–11 does the following:

- Stores the configuration information of the target system. This can be used later while performing an operation.
- Initializes all the supporting classes it uses while performing any provisioning and reconciliation operations.

4.4.1.1.2 Implementing the dispose Method

The dispose method disposes of any resources held by this Connector instance. Once the method is called, the Connector instance is discarded and can not be used. Example 4–12 illustrates how to implement the `dispose` method.

Example 4–12 dispose Method Implementation

```
/**
 * Disposes any resource used by the connector.
 */
 @Override
 public void dispose() {
//close any open FileReader or FileWriter instances.

//close connection with the target

//close connection if any with database
 }
```

4.4.1.1.3 Implementing the getConfiguration Method

The getConfiguration method returns the Configuration instance passed to the Connector when the init method was used. Example 4–13 illustrates how to implement the getConfiguration method.

Example 4–13 getConfiguration Method Implementation

```
/**
 * returns the Configuration of this connector
 */
@Override
public Configuration getConfiguration() {
    return this.flatFileConfig;
}
```

Note: Sometimes, components must be able to access the Configuration instance after initialization. This is supported by the accessor method getConfiguration().

4.4.1.2 org.identityconnectors.framework.spi.Configuration

The implementation of this interface encapsulates the configuration of a connector. Configuration implementation includes all the necessary information of the target system, which is used by the Connector implementation to connect to the target system and perform various reconciliation and provisioning operations. The implementation should have a default Constructor with setters and getters defined for its properties. Every property declared may not be required but if a property is required, then it should be marked required using the annotation org.identityconnectors.framework.spi.ConfigurationProperty. Configuration implementation is a Java bean and all the instance variables (mandatory or not) do have default values. For example, a string userName is used to connect to the target system and this is a mandatory attribute. This has a default value of null. When userName is a mandatory attribute, ICF expects a value to be provided by Oracle Identity Manager. In other words, Oracle Identity Manager cannot miss out this parameter. If missed, then the connector throws ConfigurationException.

The implementation should check that all required properties are available and validated before passing itself to the Connector. The interface provides a validate method for this purpose. For example, there are three mandatory configuration parameters, such as the IP address of the target, the username to connect to the target, and the password for the user. The validate method implementation can check for non-NULL values and valid IP address by using regex.

Note: ICF also provides a convenient base class org.identityconnectors.framework.spi.AbstractConfiguration for configuration objects to extend.

Example 4–14 Configuration Implementation

```
/**
 * Configuration implementation for the flat file connector.
 */
public class FlatFileConfigurationImpl extends AbstractConfiguration{
```

The following sections contain information and sample code that illustrates how you might implement the Configuration methods.

The Configuration implementation must provide implementation for the following methods:

- The validate() Method
- The setConnectorMessages() Method
- The getConnectorMessages() Method

4.4.1.2.1 The validate() Method

The validate method checks that the values of all required properties are set. It also validates that all values of configuration properties are valid. In other words, it validates that all values of the configuration properties are in the expected range and have the expected format. If the configuration is not valid, then the implementations generate the most specific RuntimeException available. When no specific exception is available, the implementations can throw ConfigurationException. Example 4–15 illustrates how to implement the validate method.

Example 4–15 validate Method Implementation

```
@Override
    public void validate() {
        // Validate if file exists and is usable
        boolean validFile = (this.storeFile.exists() &&
                this.storeFile.canRead() &&
                this.storeFile.canWrite() &&
                this.storeFile.isFile());
        if (!validFile)
           throw new ConfigurationException("User store file not valid");
        FlatFileIOFactory.getInstance(this);
    }
```

Here, if the target flat file provided is valid or not is checked, such as is a file, is writeable, is readable. If not valid, then an exception is generated.

Implementations of the validate method should NOT connect to the target system to validate the properties.

> **Note:** This implementation depends on an instance variable (private File storeFile) and a supporting class (FlatFileIOFactory). A complete implementation is illustrated in "Developing a Flat File Connector" on page 5-1.

4.4.1.2.2 The setConnectorMessages() Method

The setConnectorMessages method sets the org.identityconnectors.framework.common.objects.ConnectorMessages message catalog instance, allowing the Connector to localize messages. Example 4–16 illustrates the setConnectorMessages method definition.

Example 4–16 setConnectorMessages Method Definition

```
public final void setConnectorMessages(ConnectorMessages messages) {
_connectorMessages = messages;
}
```

4.4.1.2.3 The getConnectorMessages() Method

The getConnectorMessages method returns the ConnectorMessages set by the setConnectorMessages method. Example 4–17 illustrates the getConnectorMessages method definition.

Example 4–17 getConnectorMessages Method Definition

```
public final ConnectorMessages getConnectorMessages() {
return _connectorMessages;
}
```

4.4.2 Implementing the Feature-based Interfaces

The following sections contain information on the interfaces used to enable identity connector pooling and attribute normalizing.

- org.identityconnectors.framework.spi.PoolableConnector
- org.identityconnectors.framework.spi.AttributeNormalizer

4.4.2.1 org.identityconnectors.framework.spi.PoolableConnector

Connection pooling by ICF is a feature provided by the ICF in which the framework maintains a pool of connector instances and uses them while performing provisioning and reconciliation operations. Connectors can make use of pooling by implementing the PoolableConnector interface instead of plain Connector interface. To make use of this feature, implement the PoolableConnector interface. If you implement the Connector interface, then ICF creates a new connector instance for every operation, creates a new connection with the target, completes the provisioning/reconciliation operation, removes the connection with the target system, and finally disposes this connector instance. Therefore, the advantages of implementing PoolableConnector is that a pool of configurable connector instances are maintained and are reused for many operations.

Some of configurable options are:

- Maximum connector objects in the pool that are idle and active (_maxObjects)
- Maximum connector objects that are idle (_maxIdle)
- Max time to wait if the pool is waiting for a free object to become available before failing (_maxWait)
- Minimum time to wait before evicting an idle object (_minEvictableIdleTimeMillis)
- Minimum number of idle objects (_minIdle)

These values must be set by connector API developer, and if not provided, then the following default values are used:

- _maxObjects = 10
- _maxIdle = 10
- _maxWait = 150 * 1000 ms
- _minEvictableIdleTimeMillis = 120 * 1000 ms
- _minIdle = 1

The PoolableConnector interface extends the Connector interface. It is implemented to enable identity connector pooling that ICF provides. ICF must make sure that the Connector instance is alive before being used. For this purpose, the interface provides a checkAlive method. Example 4–18 is a sample flat file PoolableConnector implementation.

Example 4–18 Flat File Poolable Connector Implementation

```
/**
 * Flat file connector implementation. This is a poolable connector
   which supports create, delete, search and update operations.
 */
@ConnectorClass
  (configurationClass=FlatFileConfigurationImpl.class,
   displayNameKey="FLAT_FILE_CONNECTOR")
public class FlatFileConnector implements PoolableConnector,
   CreateOp, DeleteOp,SearchOp<Map<String, String>>,UpdateOp{
```

To implement the PoolableConnector interface, provide an implementation of the checkAlive method along with all the methods discussed in Section 4.4.1.1, "org.identityconnectors.framework.spi.Connector." The checkAlive method determines if the Connector instance is alive and can be used for operations on the target system. checkAlive can be called often thus the developer should make sure the implementation is fast. The method should throw a specific RuntimeException (if available) when the Connector is no longer alive. Example 4–19 illustrates how to implement the checkAlive method.

Example 4–19 checkAlive Method Implementation

```
/**
* Checks if this connector is alive, if not throws a RuntimeException
*/
@Override
public void checkAlive() {
//check if the connector is still connected to target
}
```

4.4.2.2 org.identityconnectors.framework.spi.AttributeNormalizer

This interface must be implemented by a Connector that needs to normalize any attributes passed to it. A normalizer converts values to a standard form for the purpose of display, consumption, or comparison. For example, a normalizer might convert text values to a specific case, trim whitespace, or order the elements of a DN in a specific way.

The interface defines a normalizeAttribute method for this purpose. This method takes an ObjectClass and an Attribute to be normalized as arguments and returns the normalized Attribute. Attribute normalization is applied during many operations including:

- Filters that are passed to SearchOp
- Results returned from SearchOp
- Results returned from SyncOp
- Attributes passed to UpdateAttributeValuesOp
- Uids returned from UpdateAttributeValuesOp
- Attributes passed to UpdateOp
- Uids returned from UpdateOp
- Attributes passed to CreateOp
- Uids returned from CreateOp
- Uids passed to DeleteOp

Example 4–20 illustrates the normalizeAttribute method definition.

Example 4–20 normalizeAttribute Method Defintion

```
public Attribute normalizeAttribute (ObjectClass oClass, Attribute attribute) {
if (attribute instanceof Uid) {
return new Uid(LdapUtil.createUniformUid((String)newValues.get(0),
configuration.getSuffix()));
}
}
```

4.4.3 Implementing the Operation Interfaces

Each operation interface defines an action that the Connector may perform on a target system, if supported by it. The operation interfaces belong to the org.identityconnectors.framework.spi.operations package. The names of these operation interfaces are listed below, but subsequent sections elaborate on each interface:

- AuthenticateOp
- CreateOp
- DeleteOp
- ResolveUsernameOp
- SchemaOp
- ScriptOnConnectorOp
- ScriptOnResourceOp
- SearchOp<T>SyncOp
- TestOp
- UpdateAttributeValuesOp
- UpdateOp

The following sections contain more information on some of these operations.

- Implementing the SchemaOp Interface
- Implementing the CreateOp Interface
- Implementing the DeleteOp Interface
- Implementing the SearchOp Interface
- Implementing the UpdateOp Interface

4.4.3.1 Implementing the SchemaOp Interface

The SchemaOp interface is implemented to allow the connector to describe the objects it can handle on the target system. The schema that a connector returns describes the object-classes that it exposes for management. Each object-class has a name, a description, and a set of attribute definitions. Each attribute definition has a name, a syntax, and certain flags that describe its properties, such as multi-valued, single-valued, readable, or writeable.

The schema that a connector returns describes the attributes of each type of object that the connector exposes. Sometimes, this requires translation from an internal representation to this Schema format. In other instances, the Schema presents as an attribute; something that is natively available only via calls to the target API. Irrespective of how the SPI implementation accomplishes the mapping between the native representation and the corresponding ConnectorObject, the Schema provides the metadata that describes what a client can expect to find in a ConnectorObject of each type, which is objectClass.

To implement this interface, provide an implementation for the schema method as defined in Example 4–21.

Example 4–21 schema Method Signature

```
public Schema schema
```

The implementation should return the schema containing the types of objects that this identity connector supports.

Example 4–22 schema Method Implementation

```
@Override
  public Schema schema() {
      SchemaBuilder flatFileSchemaBldr = new SchemaBuilder(this.getClass());
      Set<AttributeInfo> attrInfos = new HashSet<AttributeInfo>();
      for (String fieldName : flatFileMetadata.getOrderedTextFieldNames()) {
          AttributeInfoBuilder attrBuilder = new AttributeInfoBuilder();
          attrBuilder.setName(fieldName);
          attrBuilder.setCreateable(true);
          attrBuilder.setUpdateable(true);
          attrInfos.add(attrBuilder.build());
      }

// Supported class and attributes
      flatFileSchemaBldr.defineObjectClass
        (ObjectClass.ACCOUNT.getDisplayNameKey(), attrInfos);
      return flatFileSchemaBldr.build();
  }
```

Note: The Uid should not appear in the returned schema.

4.4.3.2 Implementing the CreateOp Interface

The CreateOp interface is implemented to enable creating objects on the target system. To implement this interface, provide an implementation of the create() method, as shown in Example 4–23.

Example 4–23 create Method Signature

```
public Uid create
  (ObjectClass objectClass, Set<Attribute> attributes,
   OperationOptions options)
```

This method takes an ObjectClass (for example, account or group), a set object attributes, and operation options. The implementation creates an object on the target system by using passed object attributes and object type defined by ObjectClass. The ObjectClass argument specifies the class of object to create. The class of object to be created is one of the inputs to the create operation. ObjectClass is the first argument to the create() method, as shown in Example 4–24.

Example 4–24 create Method Implementation

```
@Override
    public Uid create(ObjectClass arg0, Set<Attribute> attrs,
            OperationOptions ops) {

        System.out.println("Creating user account " + attrs);
        assertUserObjectClass(arg0);
        try {
            FlatFileUserAccount accountRecord = new FlatFileUserAccount(attrs);
        // Assert uid is there
            assertUidPresence(accountRecord);

        // Create the user
            this.flatFileWriter.addAccount(accountRecord);

        // Return uid
```

```
            String uniqueAttrField = this.flatFileConfig
                    .getUniqueAttributeName();
            String uniqueAttrVal = accountRecord
                    .getAttributeValue(uniqueAttrField);
            System.out.println("User " + uniqueAttrVal + " created");

            return new Uid(uniqueAttrVal);
        } catch (Exception ex) {

        // If account exists
            if (ex.getMessage().contains("exists"))
                throw new AlreadyExistsException(ex);

        // For all other causes
            System.out.println("Error in create " + ex.getMessage());
            throw ConnectorException.wrap(ex);
        }
    }
```

If the operation is successful, Uid instance representing object identifier on the target system is supposed to be created and returned. The caller can then use the Uid to refer to the created object.

4.4.3.3 Implementing the DeleteOp Interface

The DeleteOp interface is implemented to enable deleting objects from the target system. To implement this interface, provide an implementation for the delete method as defined in Example 4–25.

Example 4–25 delete Method Signature

```
public void delete
   (ObjectClass objectClass, Uid uid, OperationOptions options)
```

This method takes an ObjectClass (for example, account or group), the Uid of the object being deleted from the target system, and operation options. The implementation deletes the object identified by the provided Uid from the target system. if the object does not exist on the target system, then an org.identityconnectors.framework.common.exceptions.UnknownUidException is generated. Example 4–26 illustrates how to implement the delete method.

Example 4–26 delete Method Implementation

```
@Override
    public void delete(ObjectClass arg0, Uid arg1, OperationOptions arg2) {
        final String uidVal = arg1.getUidValue();
        this.flatFileWriter.deleteAccount(uidVal);
        log.ok("Account {0} deleted", uidVal);
    }
```

Note: If the delete operation fails, then ICF generates subclasses of RuntimeException. See *Oracle Fusion Middleware Java API Reference for Identity Connector Framework* for details.

4.4.3.4 Implementing the SearchOp Interface

The SearchOp interface is implemented to enable searching objects on the target system. Here, the search operation consists of:

- Creation of a native filter to implement search conditions that are specified generically.
- Executing the actual query.

Implementing these methods in the SPI allows the API to support search. The API performs (by post-processing the result) any filtering that the connector does not perform, for example, by translating any specified filter conditions into native search conditions.

To implement this interface, provide an implementation for the createFilterTranslator and executeQuery methods as documented in the following sections.

- Implementing the createFilterTranslator Method
- Implementing the executeQuery Method

4.4.3.4.1 Implementing the createFilterTranslator Method

The createFilterTranslator method returns an instance of implementation of FilterTranslator, which converts the ICF Filter object passed to it from the API side into a native query. Following the conversion, ICF passes the query to the executeQuery method. Example 4–27 illustrates the createFilterTranslator method definition.

Example 4–27 createFilterTranslator Method Signature

```
public FilterTranslator createFilterTranslator
   (ObjectClass oClass, OperationsOptions options)
```

Note: The return value should not be null.

Example 4–28 illustrates an implementation of the createFilterTranslator method.

Example 4–28 createFilterTranslator Method Implementation

```
@Override
public FilterTranslator<Map<String, String>> createFilterTranslator
  (ObjectClass arg0, OperationOptions arg1) {
   return new ContainsAllValuesImpl() {
 };
}
```

This example supports only a single type of search predicate, which is ContainsAllValues. See "Implementation of AbstractFilterTranslator<T>" on page 5-8 for an example of an implementation of ContainsAllValuesImpl. The implementation of ContainsAllValues translates into native form a condition of the form: Attribute A contains all of the values V(1), V(2) ... V(N).

For information on the org.identityconnectors.framework.common.objects.filter.FilterTranslator, see "Common Classes" on page 4-21.

4.4.3.4.2 Implementing the executeQuery Method

The executeQuery method is called for every query produced by the FilterTranslator implementation (as documented in "Implementing the createFilterTranslator Method" on page 4-19). It takes an ObjectClass (for example, account or group), the query, an instance of ResultsHandler used as a callback to handle found objects, and operation options, as illustrated in Example 4–29.

Example 4–29 executeQuery Method Signature

```
public void executeQuery
   (ObjectClass oClass, T query,
    ResultsHandler handler, OperationOptions options)
```

The implementation of the executeQuery method searches for the target objects by using the passed query, creates instances of ConnectorObject for each target object found, and uses ResultsHandler to handle ConnectorObjects. ConnectorObject is ICF representation of target resource object. It contains information such as ObjectClass, Uid, Name, and Set of Attributes. ConnectorObject is central to search. executeQuery streams ConnectorObjects into the ResultsHandler, and therefore, to the client. Example 4–30 illustrates how to implement the exectueQuery method.

Example 4–30 executeQuery Method Implementation

```
@Override
    public void executeQuery(ObjectClass objectClass,
            Map<String, String> matchSet, ResultsHandler resultHandler,
            OperationOptions ops) {


// searches the flat file for accounts which fulfil the condition 'matchSet'
created by FilterTranslator
     Iterator<FlatFileUserAccount> userAccountIterator = this.flatFileParser
              .getAccountIterator(matchSet);

boolean handleMore = true;
     while (userAccountIterator.hasNext() && handleMore) {
          FlatFileUserAccount userAcc = userAccountIterator.next();
          ConnectorObject userAccObject = convertToConnectorObject(userAcc);
          // Let the client handle the result by doing callback
     handleMore = resultHandler.handle(userAccObject);
     }
     while (userAccountIterator.hasNext()) {
          FlatFileUserAccount userAcc = userAccountIterator.next();
          ConnectorObject userAccObject = convertToConnectorObject(userAcc);
            if (!resultHandler.handle(userAccObject)) {
                System.out.println("Not able to handle " + userAcc);
                break;
            }
        }
    }
```

4.4.3.5 Implementing the UpdateOp Interface

The UpdateOp interface is implemented to enable updating objects on the target system. To implement this interface, provide an implementation of the update method as defined in Example 4–31.

Example 4–31 update Method Signature

```
public Uid update(ObjectClass oClass, Uid uid,
   Set<Attribute> attributes, OperationOptions options)
```

This method takes an ObjectClass (for example, account or group), Uid of the object being updated, a set of object attributes being updated, and operation options. The implementation updates the object on the target system identified by the Uid with the new values of attributes. If the object identified by the Uid does not exist on the target system, then an UnknowUidException is generated. Example 4–32 illustrates how to implement the update method.

Example 4–32 update Method Implementation

```
@Override
    public Uid update(ObjectClass arg0, Uid arg1,
       Set<Attribute> arg2, OperationOptions arg3) {
         String accountIdentifier = arg1.getUidValue();
    // Fetch the account
       FlatFileUserAccount accountToBeUpdated = this.flatFileParser
              .getAccount(accountIdentifier);

   // Update
        accountToBeUpdated.updateAttributes(arg2);
        this.flatFileWriter
              .modifyAccount(accountIdentifier, accountToBeUpdated);
        log.ok("Account {0} updated", accountIdentifier);

   // Return new uid
        String newAccountIdentifier = accountToBeUpdated
              .getAttributeValue
                 (this.flatFileConfig.getUniqueAttributeName());
        return new Uid(newAccountIdentifier);
    }
```

4.4.4 Common Classes

There are many ICF classes mentioned in the previous sections. The most important classes are:

- org.identityconnectors.framework.common.objects.Attribute

  An Attribute is a named collection of values within a target system object. A target system object may have many attributes and each may have many values. In its simplest form, an Attribute can be considered a name-value pair of a target system object. Empty and null values are supported. The developer should use org.identityconnectors.framework.common.objects.AttributeBuilder to construct Attribute instances.

  Note: All attributes are syntactically multivalued in this model. A particular attribute being singlevalued is only a semantic restriction.

- org.identityconnectors.framework.common.objects.Uid

  A single-valued Attribute (Uid is a subclass of Attribute) that represents the unique identifier of an object on the target resource. Ideally, it should be immutable.

Note: A singlevalued attribute is particularly relevant to UID being a unique identifier.

- org.identityconnectors.framework.common.objects.ObjectClass

  An ObjectClass defines the type of the object on the target system. Account, group, or organization are examples of such types. ICF defines predefined ObjectClasses for account (ObjectClass.ACCOUNT) and group (ObjectClass.GROUP).

- org.identityconnectors.framework.common.objects.ConnectorObject

  A ConnectorObject represents an object (for example, an account or group) on the target system. The developer must use org.identityconnectors.framework.common.objects.ConnectorObjectBuilder to construct a ConnectorObject.

- org.identityconnectors.common.security.GuardedString

  A guarded string is a secure String implementation which solves the problem of storing passwords in memory in a plain String format. Passwords are stored as Bytes in an encrypted format. The encryption key will be randomly generated.

- org.identityconnectors.framework.common.objects.filter.FilterTranslator

  A FilterTranslater object is responsible for converting all the filters specified on the API side of the ICF into native queries during a search operation. ICF Filters support both search predicates and logical operators:

  – Search predicates match objects based on the values of a specified attribute. For example, an EqualsFilter returns true when at least one value of an attribute is equal to a specified value.

  – Logical operators AND and OR join search predicates to build complex expressions. For example, an expression of the form "A AND B" is true only if both A and B are true. An expression of the form "A OR B" is true if at least one of A or B is true.

  The ICF provides the AbstractFilterTranslator<T> base class to make search implementation easier. A FilterTranslator sub class should override the following whenever possible.

  – createAndExpression(T, T)

  – createOrExpression(T, T)

  – createContainsExpression(ContainsFilter, boolean)

  – createEndsWithExpression(EndsWithFilter, boolean)

  – createEqualsExpression(EqualsFilter, boolean)

  – createGreaterThanExpression(GreaterThanFilter, boolean)

  – createGreaterThanOrEqualExpression(GreaterThanOrEqualFilter, boolean)

  – createStartsWithExpression(StartsWithFilter, boolean)

  – createContainsAllValuesExpression(ContainsAllValuesFilter, boolean)

  For more information see Section 4.4.3.4, "Implementing the SearchOp Interface."

- org.identityconnectors.framework.common.objects.ResultsHandler

  This is a callback interface for operations returning one or more results. The sub class should provide an implementation to the handle method whereas the caller can decide what to do with the results. Currently, this is used only by the SearchOp interface. For more information, see Section 4.4.3.4, "Implementing the SearchOp Interface."

4.5 Extending an Identity Connector Bundle

An identity connector bundle is the specific implementation for a particular target system. The bundle is a Java archive (JAR) that contains all the files required by the identity connector to connect to the target system and perform operations. It also has special attributes (defined in the MANIFEST file) that are recognized by the ICF. These are:

- **ConnectorBundle-FrameworkVersion** is the minimum version of the ICF required for this identity connector bundle to work. Newer ICF versions will be backwards compatible.
- **ConnectorBundle-Name** is the unique name for this identity connector bundle; it is generally the package name.
- **ConnectorBundle-Version** is the version of this bundle. Within a given deployment of Oracle Identity Manager, the ConnectorBundle-Name and ConnectorBundle-Version combination should be unique.

You extend an identity connector bundle, for example, to reuse common code. The AbtractDatabaseConnector is a good example, because different types of connectors can reuse the same basic logic that accesses database tables using JDBC. A connector for database tables might share this common code with a connector for Oracle Database users, a connector for IBM DB2 database users, and a connector for MySQL users.

A given Connector can be extended by adding the extended bundle to the /lib directory of a new bundle and creating a new class that subclasses the target class. This can be illustrated with the AbstractDatabaseConnector bundle. The common logic would be in a common bundle as follows:

Note: You do not extend the original bundle. Instead, you extend the connector by embedding the original bundle in a new bundle that wraps the original bundle.

- META-INF/MANIFEST.MF
  - ConnectorBundle-FrameworkVersion: 1.0
  - ConnectorBundle-Name: org.identityconnectors.database.common
  - ConnectorBundle-Version: 1.0
- org.identityconnectors/database/common/AbstractDatabaseConnector.class

Note: This identity connector would not have a @ConnectorClass annotation.

- org/identityconnectors/database/common/* (other common source files)
- lib/

There would be as many database (resource) specific bundles as needed. For example:

- META-INF/MANIFEST.MF
  - ConnectorBundle-FrameworkVersion: 1.0
  - ConnectorBundle-Name: org.identityconnectors.database.mysql
  - ConnectorBundle-Version: 1.0
- org/identityconnectors/database/mysql/MySQLConnector.class (subclass of AbstractDatabaseConnector)

Note: This identity connector would have a @ConnectorClass annotation.

- org/identityconnectors/database/mysql/* (other MySQL source files)
- lib/org.identityconnectors.database.common-1.0.jar (parent bundle described above)
- lib/* (specific database drivers and libraries as needed)

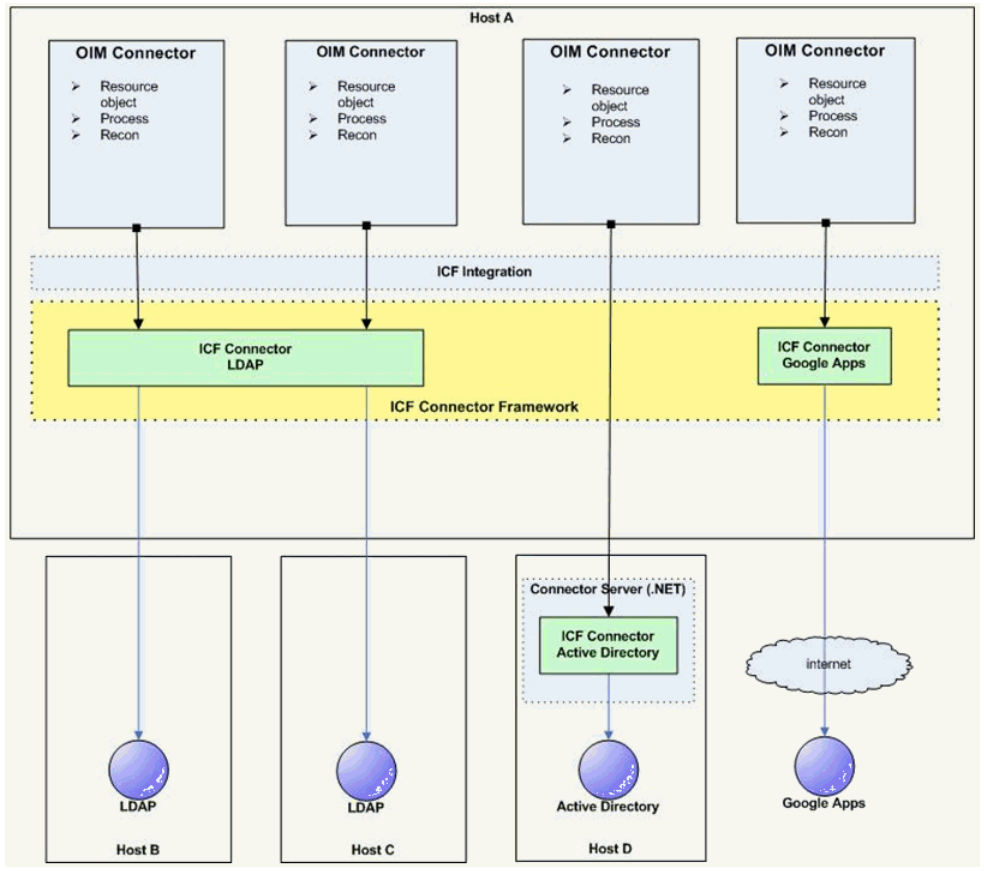
4.6 Using an Identity Connector Server

An identity connector server is required when an identity connector bundle is not directly executed within your application. By using one or more identity connector servers, the ICF architecture permits your application to communicate with externally deployed identity connector bundles. Identity connector servers are available for Java™ and Microsoft .NET Framework applications.

A single connector server can support multiple ICF connectors, and these ICF connectors may be of different connector types. A single ICF connector can be used to communicate with multiple targets.

Figure 4–6 shows how Oracle Identity Manager connectors integrate with resources via ICF connectors:

Figure 4–6 ICF Connectors and Connector Server



In Figure 4–6:

- Oracle Identity Manager connectors do not directly interact with the target resource. Instead, the create, read, update, delete, and query (CRUDQ) operations are performed via the appropriate ICF connector.
- A single ICF Connector can be used to connect to multiple resources of the same resource type. In Figure 4–6, an ICF Connector for LDAP is used to connect to a local LDAP resource, as well as being used to connect to a remote LDAP resource.
- The .NET Connector Server is used to deploy .NET ICF Connectors on the target host. An Active Directory resource is connected in this manner.
- An ICF Connector for Google Apps provides a connection to Google Apps across the Internet.
- While not shown in the diagram, a Connector Server can support multiple ICF Connectors of different resource types.

The types of connector servers are described in the following sections:

- Using the Java Connector Server
- Using the .NET Connector Server

Tip: Get the following information (defined during installation) for use during either Connector Server configuration.

- Host name and IP address
- Connector Server port
- Connector Server key
- SSL enabled

4.6.1 Using the Java Connector Server

A Java Connector Server is used when you do not want to execute a Java Connector Bundle in the same Java Virtual Machine (JVM) as the application. This deployment may be beneficial in terms of performance as the bundle works faster when deployed on the same host as the managed target system. In addition, use Java Connector Server to eliminate possibility of an application JVM crash because of faulty JNI-based connector.

Using the Java connector server is described in the following sections:

- Installing and Configuring a Java Connector Server
- Running the Java Connector Server on Microsoft Windows
- Running the Java Connector Server on Solaris and Linux
- Installing an Identity Connector in a Java Connector Server
- Using SSL to Communicate with a Connector Server

4.6.1.1 Installing and Configuring a Java Connector Server

To install and configure the Java Connector Server:

1. Create a new directory on the computer on which you want to install the Java Connector Server. In this section, *CONNECTOR_SERVER_HOME* represents this directory.
2. Unzip the Java Connector Server package in your new directory from Step 1. Java Connector Server is available for download in the Oracle Technology Network Web site at the following URL:

   `http://www.oracle.com/technetwork/index.html`
3. In the ConnectorServer.properties file in the conf/ directory, set the properties as required by your deployment. Table 4–1 lists the properties in the ConnectorServer.properties file:

Table 4–1 Properties in the ConnectorServer.properties File

Property	Description
connectorserver.port	This is a common property for denoting both SSL and non-SSL connector server port. If the **connectorserver.usessl** property is set to `true`, then the connector server listens on a secure channel using the same port. The default SSL and non-SSL port number is `8759`. To change the default connector server port (SSL or non-SSL), update the connectorserver.port property with the new port number in the ConnectorServer.properties file.
connectorserver.bundleDir	Directory where the connector bundles are deployed. The default value is `bundles`.

Table 4–1 (Cont.) Properties in the ConnectorServer.properties File

Property	Description
connectorserver.libDir	Directory in which to place dependent libraries. The default value is `lib`.
connectorserver.usessl	If set to true, the Java Connector Server uses SSL for secure communication. The default value is `false`. If you specify true, then use the following options on the command line when you start the Java Connector Server: ■ `-Djavax.net.ssl.keyStore` ■ `-Djavax.net.ssl.keyStoreType` (optional) ■ `-Djavax.net.ssl.keyStorePassword`
connectorserver.ifaddress	Bind address. To set this property, uncomment it in the file, if required. The bind address can be useful if there are more NICs installed on the computer.
connectorserver.key	Java Connector Server key.

4. Set the properties in the ConnectorServer.properties file, as follows:
   - To set connectorserver.key, run the Java Connector Server with the /setKey option. See "Running the Java Connector Server on Microsoft Windows" on page 4-27 and "Running the Java Connector Server on Solaris and Linux" on page 4-28 for more information.
   - For all other properties, edit the ConnectorServer.properties file manually.
5. The conf directory also contains the logging.properties file, which you can edit if required by your deployment.

4.6.1.2 Running the Java Connector Server on Microsoft Windows

To run the Java Connector Server on Microsoft Windows, use the ConnectorServer.bat script, as follows:

1. Make sure that you have set the properties required by your deployment in the ConnectorServer.properties file, as described in "Installing and Configuring a Java Connector Server" on page 4-26.
2. Change to the *CONNECTOR_SERVER_HOME*\bin directory and find the ConnectorServer.bat script.

   Table 4–2 lists the options supported by the ConnectorServer.bat script:

Table 4–2 Options Supported by the ConnectorServer.bat Script

Option	Description
/install [serviceName] ["-J java-option"]	Installs the Java Connector Server as a Microsoft Windows service. Optionally, you can specify a service name and Java options. If you do not specify a service name, then the default name is ConnectorServerJava.

Table 4–2 (Cont.) Options Supported by the ConnectorServer.bat Script

Option	Description
/run ["-J java-option"]	Runs the Java Connector Server from the console. Optionally, you can specify Java options. For example, to run the Java Connector Server with SSL: `ConnectorServer.bat /run "-J-Djavax.net.ssl.keyStore=mykeystore.jks" "-J-Djavax.net.ssl.keyStorePassword=password"`
/setKey [key]	Sets the Java Connector Server key. The ConnectorServer.bat script stores the hashed value of the key in the connectorserver.key property in the ConnectorServer.properties file.
/uninstall [serviceName]	Uninstalls the Java Connector Server. If you do not specify a service name, then the script uninstalls the ConnectorServerJava service.

3. If you need to stop the Java Connector Server, then stop the respective Microsoft Windows service.

4.6.1.3 Running the Java Connector Server on Solaris and Linux

To run the Java Connector Server on Solaris and Linux, use the connectorserver.sh script, as follows:

1. Make sure that you have set the properties required by your deployment in the ConnectorServer.properties file, as described in "Installing and Configuring a Java Connector Server" on page 4-26.
2. Change to the CONNECTOR_SERVER_HOME/bin directory.
3. Use the `chmod` command to set the permissions to make the connectorserver.sh script executable.
4. Run the connectorserver.sh script.

   Table 4–3 lists the options supported by the connectorserver.sh script:

Table 4–3 Options Supported by the connectorserver.sh Script

Option	Description
/run [-Jjava-option]	Runs the Java Connector Server in the console. Optionally, you can specify one or more Java options. For example, to run the Java Connector Server with SSL: `./connectorserver.sh /run -J-Djavax.net.ssl.keyStore=mykeystore.jks -J-Djavax.net.ssl.keyStorePassword=password`
/start [-Jjava-option]	Runs the Java Connector Server in the background. Optionally, you can specify one or more Java options.
/stop	Stops the Java Connector Server, waiting up to 5 seconds for the process to end.
/stop n	Stops the Java Connector Server, waiting up to n seconds for the process to end.
/stop -force	Stops the Java Connector Server. Waits up to 5 seconds, and then uses the `kill -KILL` command if the process is still running.

Table 4–3 (Cont.) Options Supported by the connectorserver.sh Script

Option	Description
/stop n -force	Stops the Java Connector Server. Waits up to n seconds, and then uses the `kill -KILL` command if the process is still running.
/setKey key	Sets the Java Connector Server key. The connectorserver.sh script stores the hashed value of the key in the connectorserver.key property in the ConnectorServer.properties file.

4.6.1.4 Installing an Identity Connector in a Java Connector Server

This section contains the procedures to deploy a Java Connector Bundle in a Java Connector Server.

1. Change to the bundles directory in your Java Connector Server directory.
2. Copy the Java Connector Bundle JAR to the bundles directory.
3. Add any applicable third party JAR files required by the identity connector to the lib directory.
4. Restart the Java Connector Server.

4.6.1.5 Using SSL to Communicate with a Connector Server

Follow these steps to communicate with a Connector Server using Secure Sockets Layer (SSL).

1. Deploy an SSL certificate to the Connector Server's system.
2. Configure your Connector Server to provide SSL sockets.
3. Configure your application to communicate with the Connector Server using SSL.

   Refer to the target system's manual for specific notes on configuring connections to identity connector servers. You will indicate to your application that an SSL connection is required when establishing a connection for each SSL-enabled connector server. Additionally, if any of the SSL certificates used by your connector servers are issued by a non-standard certificate authority, your application must be configured to respect the additional authorities. Refer to your manual for notes regarding certificate authorities.

Note: Java applications may solve the issue of non-standard certificate authorities by expecting the following Java system properties to be passed when launching the application:

- javax.net.ssl.trustStorePassword

  For example:

  ```
  -Djavax.net.ssl.trustStorePassword=changeit
  ```

- javax.net.ssl.trustStore

  For example:

  ```
  -Djavax.net.ssl.trustStore=/usr/myApp_cacerts
  ```

Alternately, the non-standard certificate authorities may be imported to the standard ${JAVA_HOME}/lib/security/cacerts directory.

4.6.2 Using the .NET Connector Server

The use of a .NET Connector Server is useful when an application is written in Java but a Connector Bundle is written using C#. Because a Java Platform, Enterprise Edition (JEE™) application cannot load C# classes, you can deploy the C# bundles under a .NET Connector Server. The Java application can then communicate with the .NET Connector Server over the network. The .NET Connector Server serves as a proxy to provide any authenticated application access to the C# bundles. The following sections contain additional information.

- Installing the .NET Connector Server
- Configuring the .NET Connector Server
- Upgrading the .NET Connector Server
- Configuring Trace Settings
- Running the .NET Connector Server
- Installing Multiple Connectors on a .NET Connector Server

4.6.2.1 Installing the .NET Connector Server

The minimum requirements to run a .NET Connector Server 12.2.1.3.0 are:

- Microsoft Windows Server 2003, 2008, or 2012
- Microsoft .NET Framework 4.5 or higher

Refer to the particular .NET identity connector documentation to determine if there are additional requirements.

To install the .NET Connector Server 12.2.1.3.0:

1. Download the Connector Server package (`Connector_Server_122130_dotnet.zip`) from the Oracle Technology Network site at the following URL:

   `http://www.oracle.com/technetwork/index.html`

2. Extract the contents of the Connector Server package and locate the `ServiceInstall-version.msi` file.

   Note: We cannot run all connector operations with a less privileged user. Therefore, we must use an admin user account (as a minimum privileged user) to run the .NET Connector Server.

3. Install the Connector Server by running the `ServiceInstall-version.msi` file and following the wizard. The wizard takes you through the installation process step-by-step. After completion, the .NET Connector Server is registered as a Windows service.

4.6.2.2 Configuring the .NET Connector Server

Common configurations include port, trace and SSL settings as well as the Connector Server key.

To configure the .NET Connector Server:

1. Start the Microsoft Services Console.

   If the .NET Connector Server is running, stop it by stopping the Windows service.

2. Go to the directory where the .NET Connector Server is installed. The default directory is, `C:\Program Files(x86)\Idenity Connector\Connector Server`.

   Run the following command from the command prompt:

   ```
   ConnectorServer.exe /setkey NEW_KEY
   ```

   In this command, `NEW_KEY` is the value for the new key. This key is required by any client that connects to this .NET Connector Server.

3. Update the settings in the .NET Connector Server configuration file (`ConnectorServer.exe.config`). These settings are in the element named `appSettings`. For example:

   ```
   <add key="connectorserver.port" value="8759"/>
   <add key="connectorserver.usessl" value="false"/>
   <add key="connectorserver.certificatestorename"
   value="ConnectorServerSSLCertificate"/>
   <add key="connectorserver.ifaddress" value="0.0.0.0"/>
   <add key="logging.proxy" value="false"/>
   <!--Possible protocol values are Tls, Tls11, Tls12 -->
   <add key="connectorserver.protocol" value="Tls">
   ```

   The most common settings you might want to change are:

   - **Port Number**: The `connectorserver.port` property is a key for denoting both SSL and non-SSL connector server port. If the `connectorserver.usessl` property is set to true, then the connector server listens on a secure channel using the same port. The default SSL and non-SSL port number is 8759. To change the default connector server port, update the `connectorserver.port` property with the new port number in the `connectorserver.exe.config` file.
   - **SSL Setting**: To use SSL, set the value of `connectorserver.usessl` to **true**, and set the value of `connectorserver.certifacatestorename` to the name of the certificate store which is having the trust certificate. Run the following command from the command prompt to create and add the target system trust certificate in certificate store:

     ```
     C:\>certutil -f -addstore tlsstore C:\ADSSLCer.cer
     Note: refer the connector guide of respective target system to generate the
     trust certificate i.e.  C:\ADSSLCer.cer
     ```

     Note: Make sure that the certificate store mentioned in the command does not already exist. The certificate store mentioned in the `ConnectorServer.exe.Config` file must have only one certificate. If there is more than one certificates, then the .NET Connector Server will not start.

     Run the following command to view the number of certificates present in the certificate store:

     ```
     C:\>certutil -viewstoreSTORE_NAME
     ```

     And update the `connectorserver.protocal` to choose the SSL communication protocol that is, TLS 1.0, TLS 1.1, or TLS 1.2.

   - **Listening Socket Bind**: To change the listening socket bind, set `connectorserver.ifaddress` to an address other than 0.0.0.0.

4. Save the following configuration information from the .NET Connector Server installation. This information must be specified while configuring the IT resource for the Connector Server:
   - Host name or IP address
   - Connector Server port
   - Connector Server key values
   - If SSL should be enabled
5. After completing all the .NET Connector Server configurations, restart the Windows service or alternately restart the .NET Connector Server by running the following command from the command line interface. Go to the directory where .NET Connector Server is installed, run the following command:

   ```
   ConnectorServer.exe /run
   ```

4.6.2.3 Upgrading the .NET Connector Server

In the 12.2.1.3.0 version of the .NET Connector Server pack, you can select the protocol for SSL communication between Oracle Identity Manager and .NET Connector Server by using the `connectorserver.protocol` property. The supported values of this property are `Tls`, `Tls11`, and `Tls12`. Here, `Tls` denotes TLS 1.0 protocol, `Tls11` denotes TLS 1.1 protocol, and `Tls12` denotes TLS 1.2 protocol. The default value of this property is `Tls`, which denotes TLS 1.0 protocol.

To upgrade the .NET Connector Server:

1. Stop the connector server service.
2. Create a backup of the directory on which Connector server is installed.
3. Download the Connector Server package (Connector_Server_122130_dotnet.zip) from the Oracle Technology Network site at the following URL:

   `http://www.oracle.com/technetwork/index.html`
4. Extract the contents of the Connector Server package and locate the `ServiceInstall-version.msi` file.

   Note: You cannot run all connector operations with a less privileged user. Therefore, you must use an admin user account (as a minimum privileged user) to run the .NET Connector Server.

   Use `ServiceInstall- version.msi` file shipped in 12.2.1.3.0 Connector Server pack to install the 12.2.1.3.0 binaries at existing Connector Server location. That is, directory `C:\Program Files (x86)\Identity Connectors\Connector Server` or the loaction where the Connector Server is installed.
5. Open `ConnectorServer.exe.config file` from both installed location and backup location. Update the following properties of `ConnectorServer.exe.config` file at installed location from `ConnectorServer.exe.config` file at backup location:

   ```
   <add key="connectorserver.port" value="8759"/>
   <add key="connectorserver.usessl" value="false"/>
   <add key="connectorserver.certificatestorename"
   value="ConnectorServerSSLCertificate"/>
   <add key="connectorserver.ifaddress" value="0.0.0.0"/>
   <add key="logging.proxy" value="false"/>
   ```

6. Set the keys for the newly installed connector server using the existing key value from command line.

   Change to the directory where the .NET Connector Server was installed. The default directory is:

   ```
   C:\Program Files (x86)\Identity Connectors\Connector Server
   ```

   Run the following command:

   ```
   ConnectorServer.exe /setkey EXISTING_KEY
   ```

   In this command, EXISTING_KEY is the value for the key. This key is required by any client that connects to this .NET Connector Server.

7. After completing all the .NET Connector Server configurations, restart the Windows service or alternately restart the .NET Connector Server by running the following command from the command line interface. Go to the directory where .NET Connector Server is installed, run the following command:

   ```
   ConnectorServer.exe /run
   ```

Note: The customization done previously are not preserved during upgrade of connector server as it is a manual upgrade. If you have any other customization in any of the files, then re-do the same from the back-up location.

4.6.2.4 Configuring Trace Settings

The Connector Server uses the standard .NET trace mechanism. Trace settings are defined in the connectorserver.exe.config configuration file. Example 4–33 illustrates how they are defined.

Example 4–33 Defined Trace Settings

```
<system.diagnostics>
  <trace autoflush="true" indentsize="4">
     <listeners>
       <remove name="Default" />
       <add name="myListener"
            type="System.Diagnostics.TextWriterTraceListener"
            initializeData="c:\connectorserver2.log"
            traceOutputOptions="DateTime">
       <filter type="System.Diagnostics.EventTypeFilter"
               initializeData="Information" />
       </add>
    </listeners>
  </trace>
</system.diagnostics>
```

The default settings are a good starting point but you may change these settings as follows.

- For less tracing, change the filter type's initializeData setting to *Warning* or *Error*.
- For more verbose logging, set the value to *Verbose* or *All*.

Caution: The amount of logging performed has a direct effect on the performance of the Connector Servers.

Any configuration changes require that the Connector Server be stopped and restarted.

Note: For more information about the tracing options, see Microsoft .NET documentation for System.Diagnostics.

4.6.2.5 Running the .NET Connector Server

The best way to run the .NET Connector Server is as a Windows Service. During installation, the Connector Server is installed as a Windows service. If this is not adequate for your environment, the Connector Server may be installed or uninstalled as a Windows Service by using the /install or /uninstall arguments at the command prompt.

To run the Connector Server interactively, open the command prompt, go to the directory where .NET Connector Server is installed, run the following command:

```
ConnectorServer.exe /run
```

4.6.2.6 Installing Multiple Connectors on a .NET Connector Server

To install new identity connectors, change to the directory where the Connector Server was installed, extract the new identity connector ZIP into it, and restart the Connector Server.

5

Developing Identity Connectors Using Java

This chapter is a tutorial that walks through the procedures necessary to develop an identity connector using the Identity Connector Framework (ICF) and the Oracle Identity Manager metadata. It includes information about important ICF classes and interfaces, the connector bundle, the connector server, and code samples for implementing a flat file identity connector and creating Oracle Identity Manager metadata for user provisioning and reconciliation processes. It contains the following sections:

5.1 Developing a Flat File Connector

The procedure for developing a flat file connector is to develop an implementation of the Configuration interface followed by the implementation of the Connector class. Before beginning, you must prepare IO representation modules for all flat file connector operations. This might include all or some of the following:

- Read the column names of the flat file and prepare metadata information.
- Add a record to the flat file with the corresponding column values separated by the specified delimiter.
- Delete a record to the flat file based on the UID value.
- Search operations on flat file.

This tutorial is focused on identity connector development, and therefore, these preparations are not discussed in detail.

Note: The following supporting classes are used for file input and output handling during identity connector operations:

- org.identityconnectors.flatfile.io.FlatFileIOFactory
- org.identityconnectors.flatfile.io.FlatFileMetadata
- org.identityconnectors.flatfile.io.FlatFileParser
- org.identityconnectors.flatfile.io.FlatFileWriter

See "Supporting Classes for File Input and Output Handling" on page 5-9 for the implementations of the input and output handling supporting classes.

To develop a flat file connector:

1. Implement the configuration class for the Flat File Connector by extending the org.identityconnectors.framework.spi.AbstractConfiguration base class.

   Example 5–1 is a sample of this. See Section 4.4.1.2, "org.identityconnectors.framework.spi.Configuration" for more information.

Example 5–1 Implementation of AbstractConfiguration

```
package org.identityconnectors.flatfile;
import java.io.File;
import org.identityconnectors.flatfile.io.FlatFileIOFactory;
import org.identityconnectors.framework.common.exceptions.ConfigurationException;
import org.identityconnectors.framework.spi.AbstractConfiguration;
import org.identityconnectors.framework.spi.ConfigurationProperty;
/**
 * Class for storing the flat file configuration
 */
public class FlatFileConfiguration extends AbstractConfiguration {
/*
 * Storage file name
 */
private File storeFile;
/*
 * Delimeter used
 */
private String textFieldDelimeter;
/*
 * Unique attribute field name
 */
private String uniqueAttributeName = "";
/*
 * Change attribute field name. Should be numeric
 */
private String changeLogAttributeName = "";

public File getStoreFile() {
return storeFile;
}

public String getTextFieldDelimeter() {
return textFieldDelimeter;
}

     public String getUniqueAttributeName() {
```

```
    return uniqueAttributeName;
}

public String getChangeLogAttributeName() {
    return changeLogAttributeName;
}

/**
 * Set the store file
 * @param storeFile
 */
@ConfigurationProperty(order = 1, helpMessageKey = "USER_ACCOUNT_STORE_HELP",
        displayMessageKey = "USER_ACCOUNT_STORE_DISPLAY")
public void setStoreFile(File storeFile) {
    this.storeFile = storeFile;
}

/**
 * Set the text field delimeter
 * @param textFieldDelimeter
 */
@ConfigurationProperty(order = 2,
        helpMessageKey = "USER_STORE_TEXT_DELIM_HELP",
        displayMessageKey = "USER_STORE_TEXT_DELIM_DISPLAY")
public void setTextFieldDelimeter(String textFieldDelimeter) {
    this.textFieldDelimeter = textFieldDelimeter;
}

/**
 * Set the field whose values will be considered as unique attributes
 * @param uniqueAttributeName
 */
@ConfigurationProperty(order = 3, helpMessageKey = "UNIQUE_ATTR_HELP",
        displayMessageKey = "UNIQUE_ATTR_DISPLAY")
public void setUniqueAttributeName(String uniqueAttributeName) {
    this.uniqueAttributeName = uniqueAttributeName;
}

/**
 * Set the field name where change number should be stored
 * @param changeLogAttributeName
 */
@ConfigurationProperty(order = 3, helpMessageKey = "CHANGELOG_ATTR_HELP",
        displayMessageKey = "CHANGELOG_ATTR_DISPLAY")
public void setChangeLogAttributeName(String changeLogAttributeName) {
    this.changeLogAttributeName = changeLogAttributeName;
}
@Override
public void validate() {

    // Validate if file exists and is usable
    boolean validFile = (this.storeFile.exists() &&
            this.storeFile.canRead() &&
            this.storeFile.canWrite() &&
            this.storeFile.isFile());

    if (!validFile)
        throw new ConfigurationException("User store file not valid");

   // Validate if there is a field on name of unique attribute field name
```

```
        // Validate if there is a field on name of change attribute field name
        FlatFileIOFactory.getInstance(this);
        // Initialization does the validation
    }


}
```

2. Create connector class for the Flat File Connector by implementing the org.identityconnectors.framework.spi.Connector interface.

   Example 5–2 implements the CreateOp, DeleteOp, SearchOp and UpdateOp interfaces and thus supports all four operations. The FlatFileMetadata, FlatFileParser and FlatFileWriter classes are supporting classes. Their implementation is not shown as they do not belong to the ICF.

Example 5–2 Implementation of PoolableConnector

```
package org.identityconnectors.flatfile;

import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Set;

import org.identityconnectors.flatfile.io.FlatFileIOFactory;
import org.identityconnectors.flatfile.io.FlatFileMetadata;
import org.identityconnectors.flatfile.io.FlatFileParser;
import org.identityconnectors.flatfile.io.FlatFileWriter;
import org.identityconnectors.framework.api.operations.GetApiOp;
import org.identityconnectors.framework.common.exceptions.AlreadyExistsException;
import org.identityconnectors.framework.common.exceptions.ConnectorException;
import org.identityconnectors.framework.common.objects.Attribute;
import org.identityconnectors.framework.common.objects.AttributeInfo;
import org.identityconnectors.framework.common.objects.AttributeInfoBuilder;
import org.identityconnectors.framework.common.objects.ConnectorObject;
import org.identityconnectors.framework.common.objects.ConnectorObjectBuilder;
import org.identityconnectors.framework.common.objects.ObjectClass;
import org.identityconnectors.framework.common.objects.OperationOptions;
import org.identityconnectors.framework.common.objects.ResultsHandler;
import org.identityconnectors.framework.common.objects.Schema;
import org.identityconnectors.framework.common.objects.SchemaBuilder;
import org.identityconnectors.framework.common.objects.Uid;
import
org.identityconnectors.framework.common.objects.filter.AbstractFilterTranslator;
import org.identityconnectors.framework.common.objects.filter.FilterTranslator;
import org.identityconnectors.framework.spi.Configuration;
import org.identityconnectors.framework.spi.ConnectorClass;
import org.identityconnectors.framework.spi.PoolableConnector;
import org.identityconnectors.framework.spi.operations.CreateOp;
import org.identityconnectors.framework.spi.operations.DeleteOp;
import org.identityconnectors.framework.spi.operations.SchemaOp;
import org.identityconnectors.framework.spi.operations.SearchOp;
import org.identityconnectors.framework.spi.operations.UpdateOp;

/**
 * The main connector class
 */
@ConnectorClass(configurationClass = FlatFileConfiguration.class, displayNameKey =
"FlatFile")
public class FlatFileConnector implements SchemaOp, CreateOp, DeleteOp,
        UpdateOp, SearchOp<Map<String, String>>, GetApiOp, PoolableConnector {

    private FlatFileConfiguration flatFileConfig;
    private FlatFileMetadata flatFileMetadata;
    private FlatFileParser flatFileParser;
    private FlatFileWriter flatFileWriter;
    private boolean alive = false;

    @Override
    public Configuration getConfiguration() {
        return this.flatFileConfig;
    }

    @Override
    public void init(Configuration config) {
        this.flatFileConfig = (FlatFileConfiguration) config;

        FlatFileIOFactory flatFileIOFactory =
             FlatFileIOFactory.getInstance(flatFileConfig);
        this.flatFileMetadata = flatFileIOFactory.getMetadataInstance();
        this.flatFileParser = flatFileIOFactory.getFileParserInstance();
        this.flatFileWriter = flatFileIOFactory.getFileWriterInstance();
        this.alive = true;
        System.out.println("init called: Initialization done");
    }

    @Override
    public void dispose() {
        this.alive = false;
    }

    @Override
    public Schema schema() {
        SchemaBuilder flatFileSchemaBldr = new SchemaBuilder(this.getClass());
        Set<AttributeInfo> attrInfos = new HashSet<AttributeInfo>();
        for (String fieldName : flatFileMetadata.getOrderedTextFieldNames()) {
            AttributeInfoBuilder attrBuilder = new AttributeInfoBuilder();
            attrBuilder.setName(fieldName);
            attrBuilder.setCreateable(true);
            attrBuilder.setUpdateable(true);
            attrInfos.add(attrBuilder.build());
        }

        // Supported class and attributes
        flatFileSchemaBldr.defineObjectClass
          (ObjectClass.ACCOUNT.getDisplayNameKey(),attrInfos);
        System.out.println("schema called: Built the schema properly");
        return flatFileSchemaBldr.build();
    }

    @Override
    public Uid create(ObjectClass arg0, Set<Attribute> attrs,
            OperationOptions ops) {

        System.out.println("Creating user account " + attrs);
        assertUserObjectClass(arg0);
        try {
            FlatFileUserAccount accountRecord = new FlatFileUserAccount(attrs);
            // Assert uid is there
            assertUidPresence(accountRecord);

            // Create the user
            this.flatFileWriter.addAccount(accountRecord);

            // Return uid
            String uniqueAttrField = this.flatFileConfig
                    .getUniqueAttributeName();
            String uniqueAttrVal = accountRecord
                    .getAttributeValue(uniqueAttrField);
            System.out.println("User " + uniqueAttrVal + " created");

            return new Uid(uniqueAttrVal);
        } catch (Exception ex) {

            // If account exists
            if (ex.getMessage().contains("exists"))
                throw new AlreadyExistsException(ex);

            // For all other causes
            System.out.println("Error in create " + ex.getMessage());
            throw ConnectorException.wrap(ex);
        }
    }

    @Override
    public void delete(ObjectClass arg0, Uid arg1, OperationOptions arg2) {
        final String uidVal = arg1.getUidValue();
        this.flatFileWriter.deleteAccount(uidVal);
        System.out.println("Account " + uidVal + " deleted");
    }

    @Override
    public Uid update(ObjectClass arg0, Uid arg1, Set<Attribute> arg2,
            OperationOptions arg3) {
        String accountIdentifier = arg1.getUidValue();
        // Fetch the account
        FlatFileUserAccount accountToBeUpdated = this.flatFileParser
                .getAccount(accountIdentifier);

        // Update
        accountToBeUpdated.updateAttributes(arg2);
        this.flatFileWriter
                .modifyAccount(accountIdentifier, accountToBeUpdated);
        System.out.println("Account " + accountIdentifier + " updated");

        // Return new uid
        String newAccountIdentifier = accountToBeUpdated
                .getAttributeValue(this.flatFileConfig.getUniqueAttributeName());
        return new Uid(newAccountIdentifier);
    }

    @Override
    public FilterTranslator<Map<String, String>> createFilterTranslator(
            ObjectClass arg0, OperationOptions arg1) {
        // TODO: Create a fine grained filter translator

        // Return a dummy object as its not applicable here.
        // All processing happens in the execute query
        return new AbstractFilterTranslator<Map<String, String>>() {
        };
    }

    @Override
    public ConnectorObject getObject(ObjectClass arg0, Uid uid,
            OperationOptions arg2) {
        // Return matching record
        String accountIdentifier = uid.getUidValue();
        FlatFileUserAccount userAcc = this.flatFileParser
                .getAccount(accountIdentifier);
        ConnectorObject userAccConnObject = convertToConnectorObject(userAcc);
        return userAccConnObject;
    }

    /*
     * (non-Javadoc)
     * This is the search implementation.
     * The Map passed as the query here, will map to all the records with
     * matching attributes.
     *
     * The record will be filtered if any of the matching attributes are not
     * found
     *
     * @see
     * org.identityconnectors.framework.spi.operations.SearchOp#executeQuery
     * (org.identityconnectors.framework.common.objects.ObjectClass,
     * java.lang.Object,
     * org.identityconnectors.framework.common.objects.ResultsHandler,
     * org.identityconnectors.framework.common.objects.OperationOptions)
     */
    @Override
    public void executeQuery(ObjectClass objectClass,
            Map<String, String> matchSet, ResultsHandler resultHandler,
            OperationOptions ops) {

    System.out.println("Inside executeQuery");

        // Iterate over the records and handle individually
        Iterator<FlatFileUserAccount> userAccountIterator = this.flatFileParser
                .getAccountIterator(matchSet);

        while (userAccountIterator.hasNext()) {
            FlatFileUserAccount userAcc = userAccountIterator.next();
            ConnectorObject userAccObject = convertToConnectorObject(userAcc);
            if (!resultHandler.handle(userAccObject)) {
                System.out.println("Not able to handle " + userAcc);
                break;
            }
        }
    }

    private void assertUserObjectClass(ObjectClass arg0) {
        if (!arg0.equals(ObjectClass.ACCOUNT))
            throw new UnsupportedOperationException(
                    "Only user account operations supported.");

    }

    private void assertUidPresence(FlatFileUserAccount accountRecord) {
        String uniqueAttrField = this.flatFileConfig.getUniqueAttributeName();
        String uniqueAttrVal = accountRecord.getAttributeValue(uniqueAttrField);

        if (uniqueAttrVal == null) {
            throw new IllegalArgumentException("Unique attribute not passed");
        }
    }

    private ConnectorObject convertToConnectorObject(FlatFileUserAccount userAcc)
{
        ConnectorObjectBuilder userObjBuilder = new ConnectorObjectBuilder();
        // Add attributes
        List<String> attributeNames = this.flatFileMetadata
                .getOrderedTextFieldNames();
        for (String attributeName : attributeNames) {
            String attributeVal = userAcc.getAttributeValue(attributeName);
            userObjBuilder.addAttribute(attributeName, attributeVal);

            if (attributeName.equals(this.flatFileConfig
                    .getUniqueAttributeName())) {
                userObjBuilder.setUid(attributeVal);
                userObjBuilder.setName(attributeVal);
            }
        }
        return userObjBuilder.build();
    }

    @Override
    public void checkAlive() {
        if (!alive)
            throw new RuntimeException("Connection not alive");
    }

}
```

3. This connector supports only the ContainsAllValuesFilter operation. Implement the ContainsAllValuesFilter operation Example 5–3 illustrates the sample implementation of org.identityconnectors.framework.common.objects.filter.AbstractFilterTranslator<T> that defines the filter operation.

Example 5–3 Implementation of AbstractFilterTranslator<T>

```
package org.identityconnectors.flatfile.filteroperations;

import java.util.HashMap;
import java.util.Map;

import org.identityconnectors.framework.common.objects.Attribute;
import
org.identityconnectors.framework.common.objects.filter.AbstractFilterTranslator;
import
org.identityconnectors.framework.common.objects.filter.ContainsAllValuesFilter;

public class ContainsAllValuesImpl extends AbstractFilterTranslator<Map<String,
String>>{
@Override
protected Map<String, String> createContainsAllValuesExpression(
ContainsAllValuesFilter filter, boolean not) {
Map<String, String> containsAllMap = new HashMap<String, String>();
Attribute attr = filter.getAttribute();
containsAllMap.put(attr.getName(), attr.getValue().get(0).toString());
return containsAllMap;
}
}
```

4. Create the connector bundle JAR. The MANIFEST.MF file must contain the following entries:
   - ConnectorBundle-FrameworkVersion
   - ConnectorBundle-Name
   - ConnectorBundle-Version

   Example 5–4 shows the contents of the MANIFEST.MF file:

Example 5–4 The MANIFEST.MF File

```
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.7.0
Created-By: 14.1-b02 (Sun Microsystems Inc.)
ConnectorBundle-FrameworkVersion: 1.0
ConnectorBundle-Name: org.identityconnectors.flatfile
ConnectorBundle-Version: 1.0
Build-Number: 609
Subversion-Revision: 4582
```

5. Update the connector bundle JAR as created in step 4. To do so:
   a. Extract the connector bundle JAR into any desired location.
   b. Create a lib directory in the directory in which you extracted the JAR.
   c. Add the dependent third-party JARs into the lib directory.
   d. JAR the entire directory.

   Note: The MANIFEST.MF file must contain the entries listed in step 4.

5.1.1 Supporting Classes for File Input and Output Handling

This section shows the implementation of the following supporting classes for file input and output handling:

- Example 5–5, "FlatFileIOFactory"
- Example 5–6, "FlatFileMetadata"
- Example 5–7, "FlatFileParser"
- Example 5–8, "FlatFileWriter"
- Example 5–9, "FlatfileLineIterator"
- Example 5–10, "FlatfileUserAccount"
- Example 5–11, "FlatfileAccountConversionHandler"
- Example 5–12, "Messages.Properties"

Example 5–5 shows the implementation of the FlatFileIOFactory supporting class:

Example 5–5 FlatFileIOFactory

```
package org.identityconnectors.flatfile.io;

import org.identityconnectors.flatfile.FlatFileConfiguration;

public class FlatFileIOFactory {

    private FlatFileMetadata flatFileMetadata;
    private FlatFileConfiguration flatFileConfig;

    /**
     * Provides instance of the factory
     * @param flatfileConfig Configuration bean for the flat file
     */
    public static FlatFileIOFactory getInstance(FlatFileConfiguration fileConfig)
{
        return new FlatFileIOFactory(fileConfig);
    }

    /**
     * Making it private to avoid public instantiation. Encouraging use of
getInstance
     * @param fileConfig
     */
    private FlatFileIOFactory(FlatFileConfiguration fileConfig) {
        this.flatFileConfig = fileConfig;
        this.flatFileMetadata = new FlatFileMetadata(flatFileConfig);
        System.out.println("Metadata set");
    }

    /**
     * Returns the metadata instance
     * @return
     */
    public FlatFileMetadata getMetadataInstance() {
        return this.flatFileMetadata;
    }

    /**
     * Returns the FlatFileParser instance
     * @return
     */
    public FlatFileParser getFileParserInstance() {
        return new FlatFileParser(this.flatFileMetadata, this.flatFileConfig);
    }

    /**
     * Returns the FlatFileWriter instance
     * @return
     */
    public FlatFileWriter getFileWriterInstance() {
        return new FlatFileWriter(this.flatFileMetadata, this.flatFileConfig);
    }
}
```

Example 5–6 shows the implementation of the FlatFileMetaData supporting class:

Example 5–6 FlatFileMetadata

```
package org.identityconnectors.flatfile.io;
```

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.StringTokenizer;

import org.identityconnectors.flatfile.FlatFileConfiguration;

/**
 * This class contains all the metadata related information Example: Ordering of
 * columns, Number of columns etc.
 *
 * @author harsh
 *
 */
public class FlatFileMetadata {

    private FlatFileConfiguration fileConfig;

    private List<String> orderedTextFieldNames;

    private String changeLogFieldName;
    private String uniqueAttributeFiledName;

    /**
     * Instantiates the class with the file configuration.
     * Making it package private to encourage instantiation from Factory class
     * @param fileConfig
     */
    FlatFileMetadata(FlatFileConfiguration fileConfig) {
        /*
         * Ideally you should not take connector specific configuration class in
         * flat file resource classes. Change if this has to go to production.
         * Probably make another configuration class for flat file with same
         * signatures.
         */
        this.fileConfig = fileConfig;

        initializeMetadata();
        validateConfigProps();
    }

    /**
     * Returns the text field names in the order of their storage
     *
     * @return
     */
    public List<String> getOrderedTextFieldNames() {
        return this.orderedTextFieldNames;
    }

    /**
     * Returns the number of columns
     */
    public int getNumberOfFields() {
        int numberOfTextFields = this.orderedTextFieldNames.size();
        return numberOfTextFields;
    }
    /**
     * Specifies if number of tokens are matching with the standard length of
metadata
     * @param countTokens
     * @return
     */
    public boolean isDifferentFromNumberOfFields(int countTokens) {
        return (getNumberOfFields() != countTokens);
    }

    /**
     * Reads the header line and sets the metadata
     */
    private void initializeMetadata() {
        // Read the file.
        File recordsStore = this.fileConfig.getStoreFile();

        try {
            BufferedReader storeFileReader = new BufferedReader(new FileReader(
                    recordsStore.getAbsolutePath()));

            // Read the header line
            String headerString = storeFileReader.readLine();

            // Tokenize the headerString
            StringTokenizer tokenizer = new StringTokenizer(headerString,
                    fileConfig.getTextFieldDelimeter());

            this.orderedTextFieldNames = new ArrayList<String>();
            while (tokenizer.hasMoreTokens()) {
                String header = tokenizer.nextToken();
                this.orderedTextFieldNames.add(header);
            }

            System.out.println("Columns read - " + this.orderedTextFieldNames);
        } catch (IOException e) {
            throw new RuntimeException("How can I read a corrupted file");
        }

        // Store the change log and unique attribute field names
        this.changeLogFieldName = fileConfig.getChangeLogAttributeName();
        this.uniqueAttributeFiledName = fileConfig.getUniqueAttributeName();
    }

    /**
     * Validate if the attribute names in config props object are present in the
     * column names
     *
     * @throws RuntimeException
     *             if validation fails
     */
    private void validateConfigProps() {
        // Check if unique attribute col name is present
        if (!this.orderedTextFieldNames.contains(this.changeLogFieldName))
            throw new RuntimeException("Change log field name "
                    + this.changeLogFieldName + " not found in the store file ");

        // Check if change col name is present
        if (!this.orderedTextFieldNames.contains(this.uniqueAttributeFiledName))
```

```
            throw new RuntimeException("Unique attribute field name "
                    + this.uniqueAttributeFiledName
                    + " not found in the store file");
    }
}
```

Example 5–7 shows the implementation of the FlatFileParser supporting class:

Example 5–7 FlatFileParser

```
package org.identityconnectors.flatfile.io;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

import org.identityconnectors.flatfile.FlatFileConfiguration;
import org.identityconnectors.flatfile.FlatFileUserAccount;
import org.identityconnectors.flatfile.utils.AccountConversionHandler;

public class FlatFileParser {

    private File recordsStore;
    private FlatFileConfiguration fileConfig;
    private FlatFileMetadata metadata;
    private AccountConversionHandler accountConverter;

    /**
     * Instantiates the parser class. Making it package private to encourage
     * instantiation from Factory class
     *
     * @param metadata
     * @param fileConfig
     */
    FlatFileParser(FlatFileMetadata metadata, FlatFileConfiguration fileConfig) {
        this.fileConfig = fileConfig;
        this.recordsStore = fileConfig.getStoreFile();
        this.accountConverter = new AccountConversionHandler(metadata,
                fileConfig);
        this.metadata = metadata;
    }

    /**
     * Returns all accounts in the file
     *
     * @return
     */
    public List<FlatFileUserAccount> getAllAccounts() {
        try {
            BufferedReader userRecordReader = new BufferedReader(
                    new FileReader(recordsStore.getAbsolutePath()));
            String recordStr;

            // Skip headers
            userRecordReader.readLine();
            // Loop over records and make list of objects
            List<FlatFileUserAccount> allAccountRecords = new
ArrayList<FlatFileUserAccount>();
            while ((recordStr = userRecordReader.readLine()) != null) {
                try {
                    FlatFileUserAccount accountRecord = accountConverter
                            .convertStringRecordToAccountObj(recordStr);
                    allAccountRecords.add(accountRecord);
                } catch (RuntimeException e) {
                    System.out.println("Invalid entry " + e.getMessage());
                }
            }
            userRecordReader.close();

            return allAccountRecords;
        } catch (IOException e) {
            throw new RuntimeException("How can I read a corrupted file");
        }
    }

    /**
     * Gets the account of matching account identifier
     *
     * @param accountIdentifier
     * @return
     */
    public FlatFileUserAccount getAccount(String accountIdentifier) {

        /*
         * I know its not right to get all account details. Don't want to focus
         * on efficiency and scalability as this is just a sample.
         */
        // Iterate over all records and check for matching account
        Map<String, String> matchSet = new HashMap<String, String>();
        matchSet.put(fileConfig.getUniqueAttributeName(), accountIdentifier);
        for (FlatFileUserAccount userRecord : getAllAccounts()) {
            if (userRecord.hasMatchingAttributes(matchSet))
                return userRecord;
        }

        // Got nothing..
        return null;
    }

    /**
     * Returns all records with matching Attributes If more than attributes are
     * passed. it will check all the attributes
     *
     * @param matchSet
     *            Checks if all provided attributes are matched
     */
    public List<FlatFileUserAccount> getAccountsByMatchedAttrs(
            Map<String, String> matchSet) {
        /*
         * I know its not right to get all account details. Don't want to focus
         * on efficiency and scalability as this is just a sample.
         */
        // Iterate over all records and check for matching account
        List<FlatFileUserAccount> matchingRecords = new
```

```
ArrayList<FlatFileUserAccount>();
        for (FlatFileUserAccount userRecord : getAllAccounts()) {
            if (userRecord.hasMatchingAttributes(matchSet))
                matchingRecords.add(userRecord);
        }

        return matchingRecords;
    }

    /**
     * Returns the records that fall after the specified change number This
     * function helps in checking the function of sync
     *
     * @param changeNumber
     *            the change number for the last search
     */
    public List<FlatFileUserAccount> getUpdatedAccounts(int changeNumber) {
        /*
         * I know its not right to get all account details. Don't want to focus
         * on efficiency and scalability as this is just a sample.
         */
        // Iterate over all records and check for matching account
        List<FlatFileUserAccount> matchingRecords = new
ArrayList<FlatFileUserAccount>();
        String changeLogAttrName = fileConfig.getChangeLogAttributeName();
        for (FlatFileUserAccount userRecord : getAllAccounts()) {
            int recordChangeNumber = userRecord
                    .getChangeNumber(changeLogAttrName);
            if (recordChangeNumber >= changeNumber)
                matchingRecords.add(userRecord);
        }
        return matchingRecords;

    }

    /**
     * Returns an iterator that iterates over the records. This is provided for
     * dynamic retrieval of records
     *
     * @param matchSet
     *            Filters the records by matching the given attributes. Use null
     *            or empty set to avoid filtering
     * @return
     */
    public Iterator<FlatFileUserAccount> getAccountIterator(
            Map<String, String> matchSet) {
        Iterator<FlatFileUserAccount> recordIterator = new FlatFileLineIterator(
                this.metadata, this.fileConfig, matchSet);

        return recordIterator;
    }

    /**
     * Gives the next change number. Logic is max of existing change numbers + 1
     * @return
     */
    public int getNextChangeNumber() {
        int maximumChangeNumber = 0;

        /*
```

```
         * I know its not right to get all account details. Don't want to focus
         * on efficiency and scalability as this is just a sample.
         */
        // Iterate over all records and check for matching account
        String changeLogAttrName = fileConfig.getChangeLogAttributeName();
        for (FlatFileUserAccount userRecord : getAllAccounts()) {
            int changeNumber = userRecord.getChangeNumber(changeLogAttrName);

            if (changeNumber >= maximumChangeNumber) {
                maximumChangeNumber = changeNumber + 1;
            }
        }
        return maximumChangeNumber;
    }
}
```

Example 5–8 shows the implementation of the FlatFileWriter supporting class:

Example 5–8 FlatFileWriter

```
package org.identityconnectors.flatfile.io;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

import org.identityconnectors.flatfile.FlatFileConfiguration;
import org.identityconnectors.flatfile.FlatFileUserAccount;
import org.identityconnectors.flatfile.utils.AccountConversionHandler;

/**
 * Class for searching operations on files
 *
 * @author Harsh
 */
public class FlatFileWriter {

    private File recordsStore;
    private FlatFileParser recordParser;
    private FlatFileConfiguration fileConfig;
    private AccountConversionHandler accountConverter;

    /**
     * Initializes the writer with the configuration Making it package private
     * to encourage use of Factory class for global instantiation
     *
     * @param metadata
     * @param fileConfig
     */
    FlatFileWriter(FlatFileMetadata metadata, FlatFileConfiguration fileConfig) {
        this.fileConfig = fileConfig;

        this.recordsStore = fileConfig.getStoreFile();
        recordParser = new FlatFileParser(metadata, fileConfig);
        accountConverter = new AccountConversionHandler(metadata, fileConfig);
    }
```

```
/**
 * Appends the user record at the end of
 *
 * @param accountRecord
 */
public void addAccount(FlatFileUserAccount accountRecord) {
    try {
        BufferedWriter userRecordWriter = new BufferedWriter(
                new FileWriter(this.recordsStore.getAbsolutePath(), true));

        // Set the latest changelog number
        int latestChangeNumber = recordParser.getNextChangeNumber();
        accountRecord.setChangeNumber(fileConfig
                .getChangeLogAttributeName(), latestChangeNumber);

        // Validate if same account id doesn't exist
        String accountUid = accountRecord.getAttributeValue(fileConfig
                .getUniqueAttributeName());
        FlatFileUserAccount accountByAccountId = recordParser
                .getAccount(accountUid);

        if (accountByAccountId != null)
            throw new RuntimeException("Account " + accountUid
                    + " already exists");

        // Put the user record in formatted way
        String userRecordAsStr = accountConverter
                .convertAccountObjToStringRecord(accountRecord);
        userRecordWriter.write("\n" + userRecordAsStr);

        // Close the output stream
        userRecordWriter.close();
    } catch (IOException e) {// Catch exception if any
        throw new RuntimeException("How can I write on a corrupted file");
    }
}

/**
 * Removes the entry for respective account identifier
 *
 * @param accountIdentifier
 */
public void deleteAccount(String accountIdentifier) {
    String blankRecord = "";
    this.modifyAccountInStore(accountIdentifier, blankRecord);
}

/**
 * Updates the entry with respective account identifier
 *
 * @param accountIdentifier
 * @param updatedAccountRecord
 * @return new accountIdentifier
 */
public String modifyAccount(String accountIdentifier,
        FlatFileUserAccount updatedAccountRecord) {

    // Frame a record string and update back to file
    int nextChangeNumber = recordParser.getNextChangeNumber();
    String changeNumberFieldName = fileConfig.getChangeLogAttributeName();
    updatedAccountRecord.setChangeNumber(changeNumberFieldName,
            nextChangeNumber);

    String newRecordAsStr = accountConverter
            .convertAccountObjToStringRecord(updatedAccountRecord);
    // Update to the file
    this.modifyAccountInStore(accountIdentifier, newRecordAsStr);

    // Return new UID
    String uniqueAttrFieldName = fileConfig.getUniqueAttributeName();
    String newAccountIdentifier = updatedAccountRecord
            .getAttributeValue(uniqueAttrFieldName);
    return newAccountIdentifier;
}

/**
 * Returns the complete flat file as string.
 *
 * @return
 */
private String getCompleteFlatFileAsStr() {
    try {
        BufferedReader userRecordReader = new BufferedReader(
                new FileReader(recordsStore.getAbsolutePath()));
        String recordStr;

        // Loop over records and make list of objects
        StringBuilder flatFileStr = new StringBuilder();
        while ((recordStr = userRecordReader.readLine()) != null) {
            if (!recordStr.isEmpty())
                flatFileStr.append(recordStr + "\n");
        }
        userRecordReader.close();

        return flatFileStr.toString();
    } catch (IOException e) {
        throw new RuntimeException("How can I read a corrupted file");
    }
}

/**
 * Updates the account with the new record. this can also be used for delete
 *
 * @param accountIdentifier
 * @param updatedRecord
 */
private void modifyAccountInStore(String accountIdentifier,
        String updatedRecord) {
    try {
        // Load the complete flat file
        String completeFlatFile = this.getCompleteFlatFileAsStr();

        // Construct the string to be removed and replace it with blank
        FlatFileUserAccount accountToBeRemoved = recordParser
                .getAccount(accountIdentifier);
        String updatableString = accountConverter
                .convertAccountObjToStringRecord(accountToBeRemoved);
        String updatedFlatFile = completeFlatFile.replaceAll(
                updatableString, updatedRecord);
            // Rewrite the file
            BufferedWriter userRecordWriter = new BufferedWriter(
                    new FileWriter(this.recordsStore.getAbsolutePath(), false));
            userRecordWriter.write(updatedFlatFile);

            /*** debug ***/
            System.out.println("Old string " + updatableString);
            System.out.println("New String" + updatedRecord);
            System.out.println("new file - " + updatedFlatFile);

            /******/
            // Close the output stream
            userRecordWriter.close();
        } catch (IOException e) {// Catch exception if any
            throw new RuntimeException("How can I write on a corrupted file");
        }
    }
}
```

Example 5–9 FlatfileLineIterator

```
package org.identityconnectors.flatfile.io;
 .
 import java.io.BufferedReader;
 import java.io.File;
 import java.io.FileReader;
 import java.io.IOException;
 import java.util.Iterator;
 import java.util.Map;
 .
 import org.identityconnectors.flatfile.FlatFileConfiguration;
 import org.identityconnectors.flatfile.FlatFileUserAccount;
 import org.identityconnectors.flatfile.utils.AccountConversionHandler;
 .
 /**
  * Iterator class to fetch the records dynamically during search operations
 This
  * is needed to prevent VM overloading when all records are stored in memory
  *
  * @author admin
  *
  */
 public class FlatFileLineIterator implements Iterator<FlatFileUserAccount> {
 .
     private File recordsStore;
     private AccountConversionHandler accountConverter;
     private FlatFileUserAccount nextRecord;
     private BufferedReader userRecordReader;
     private Map<String, String> attrConstraints;
 .
     /**
      * Making it package private to prevent global initialization
      *
      * @param metadata
      * @param fileConfig
      * @param attributeValConstraints
      *            Iterator will apply this constraint and filter the result
      */
```

```
    FlatFileLineIterator(FlatFileMetadata metadata,
            FlatFileConfiguration fileConfig,
            Map<String, String> attributeValConstraints) {
        this.recordsStore = fileConfig.getStoreFile();
        this.accountConverter = new AccountConversionHandler(metadata,
                fileConfig);
        this.attrConstraints = attributeValConstraints;
.
        initializeReader();
        this.nextRecord = readNextValidRecord();
    }
.
    private void initializeReader() {
        try {
            userRecordReader = new BufferedReader(new FileReader(recordsStore
                    .getAbsolutePath()));
.
            // Skip headers
            userRecordReader.readLine();
.
        } catch (IOException io) {
            throw new IllegalStateException("Unable to read "
                    + recordsStore.getName());
        }
    }
.
    @Override
    public boolean hasNext() {
        return (nextRecord != null);
    }
.
    @Override
    public FlatFileUserAccount next() {
        FlatFileUserAccount currentRecord = this.nextRecord;
        this.nextRecord = readNextValidRecord();
        return currentRecord;
    }
.
    @Override
    public void remove() {
        // Nothing to do here
    }
.
    /**
     * Returns next valid record. This happens after applying
     *
     * @return
     */
    private FlatFileUserAccount readNextValidRecord() {
        try {
            FlatFileUserAccount userAccObj = null;
            String recordStr;
            // Match the constraints or read next line
            do {
                System.out.println("Before record string");
                recordStr = getNextLine();
.
                // No more records ??
                if (recordStr == null)
                    return null;
.
                userAccObj = accountConverter
                        .convertStringRecordToAccountObj(recordStr);
            } while (!userAccObj.hasMatchingAttributes(attrConstraints));

            return userAccObj;
        } catch (Exception e) {
            System.out.println("Error reading record" + e.getMessage());
            e.printStackTrace();
            return null;
        }
    }
.
    private String getNextLine() throws IOException {
        String nextLine = userRecordReader.readLine();
.
        // No records ??
        if (nextLine == null) {
            this.userRecordReader.close();
            return null;
        }
.
        if (nextLine.trim().isEmpty()) {
            return getNextLine();
        }
.
        return nextLine;
    }
}
```

Example 5–10 FlatfileUserAccount

```
package org.identityconnectors.flatfile;
.
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;
.
import org.identityconnectors.framework.common.objects.Attribute;
.
/**
 * Object representing a user entity
 *
 * @author admin
 *
 */
public class FlatFileUserAccount {
.
    /*
     * Mandatory attribute names
     */
    private Set<String> mandatoryAttrNames = new HashSet<String>();
.
    /*
     * Attributes making the account
     */
    private Map<String, String> attributes = new HashMap<String, String>();
.
```

```
    /**
     * Instantiates the attribute value map
     *
     * @param mandatoryAttributeNames
     *           Names of the attributes that are necessary
     * @param attributeValMap
     *           Name value map for the attributes.
     * @throws IllegalStateException
     *            If mandatory attributes are not found in attribute val map
     */
    public FlatFileUserAccount(Set<String> mandatoryAttributeNames,
            Map<String, String> attributeValMap) {
        // Check if mandatory attribute values are passed
        Set<String> attrValuesKeySet = attributeValMap.keySet();
        if (!attrValuesKeySet.containsAll(mandatoryAttributeNames))
            throw new IllegalStateException("Mandatory attributes missing");
.
        // Initialize
        this.mandatoryAttrNames = mandatoryAttributeNames;
        this.attributes = attributeValMap;
.
    }

    /**
     * Instantiates the attribute value map.
     * Considers all attributes to be mandatory
     * @param attributeValMap
     */
    public FlatFileUserAccount(Map<String, String> attributeValMap) {
        this.mandatoryAttrNames = attributeValMap.keySet();
        this.attributes = attributeValMap;
    }

    /**
     * Instantiates the attribute value map
     * @param attrs
     */
    public FlatFileUserAccount(Set<Attribute> attrs) {
    for(Attribute attr: attrs) {
    String attrName = attr.getName();

    //Consider first value. Multivalued not supported
    String attrVal = (String) attr.getValue().get(0);
    this.attributes.put(attrName, attrVal);
    }
    }
.
    /**
     * Updates the set of attributes. If new attributes present, they are
added,
     * If old attributes are present in the parameter set, values are updated
     *
     * @param updatedAttributeValMap
     */
    public void updateAttributes(Map<String, String> updatedAttributeValMap)
{
        this.attributes.putAll(updatedAttributeValMap);
    }

    /**
```

```
     * Updates the set of attributes.
     * @param upatedAttributes
     */
    public void updateAttributes(Set<Attribute> upatedAttributes) {
    Map<String, String> updatedAttributeValMap = new HashMap<String,
String>();
    for(Attribute attr: upatedAttributes) {
    String attrName = attr.getName();

    //Consider first value. Multivalued not supported
    String attrVal = (String) attr.getValue().get(0);
    updatedAttributeValMap.put(attrName, attrVal);
    }
    this.attributes.putAll(updatedAttributeValMap);
    }
.
    /**
     * Deletes the attributes with given name.
     *
     * @param attributeKeys
     *            Set of the attribute names that are needed
     * @throws UnsupportedOperationException
     *             if delete for mandatory attributes is attempted
     */
    public void deleteAttributes(Set<String> attributeKeys) {
        // Check if mandatory attributes are not there.
        for (String attrKey : attributeKeys) {
            if (this.mandatoryAttrNames.contains(attrKey))
                throw new UnsupportedOperationException(
                        "Delete for mandatory attributes not supported. Try
update");
            // Not deleting here as it might result inconsistent
        }
        // Remove the attributes
        for (String attrKey : attributeKeys) {
            this.attributes.remove(attrKey);
        }
    }
.
    /**
     * Gets the attribute of a given name
     *
     * @param attributeName
     * @return
     * @throws IllegalArgumentException
     *             if attribute is not there for a given name
     */
    public String getAttributeValue(String attributeName) {
        return this.attributes.get(attributeName);
    }
.
    /**
     * Returns the current set of attributes
     *
     * @return
     */
    public Map<String, String> getAllAttributes() {
        return this.attributes;
    }
.
```

```
    /**
     * Returns true if all passed attributes are matching for this object
     *
     * @param attrValMap
     * @return
     */
    public boolean hasMatchingAttributes(Map<String, String> attrValMap) {
        boolean noFilterSupplied = (attrValMap == null )||
(attrValMap.isEmpty());
        if (noFilterSupplied)
            // No filter. Everything matches
            return true;

        // Iterate to match attributes one by one
        Set<String> keySet = attrValMap.keySet();
        for (String attrName : keySet) {
            String objAttrVal = this.attributes.get(attrName);
            String passedValue = attrValMap.get(attrName);
.
            if (!objAttrVal.equals(passedValue))
                // This attribute is not same
                return false;
        }
.
        // All attributes are same
        return true;
    }
.
    /**
     * Returns the change log number
     *
     * @param changeLogAttrName
     *            attribute representing the number
     * @return
     */
    public int getChangeNumber(String changeLogAttrName) {
        String changeNumStr = this.attributes.get(changeLogAttrName);
        int changeNumber = 0;
.
        try {
            changeNumber = Integer.parseInt(changeNumStr);
        } catch (Exception e) {
            System.out.println("Not a valid change log number "
                    + changeLogAttrName + " :" + changeNumStr);
        }
.
        return changeNumber;
    }

    /**
     * Sets the given attribute with a new value
     * @param attrName
     * @param attrVal
     */
    public void setAttribute(String attrName, String attrVal) {
        this.attributes.put(attrName, attrVal);
    }

    /**
     * Updates the changelog number
     * @param changeLogAttrName
     * @param newChangeNumber
     */
    public void setChangeNumber(String changeLogAttrName, int
newChangeNumber) {
        String changeNumberValStr = "" + newChangeNumber;
        this.attributes.put(changeLogAttrName, changeNumberValStr);
    }
.
    @Override
    public String toString() {
        // Just print the attributes
        return this.attributes.toString();
    }
.
}
```

Example 5–11 FlatfileAccountConversionHandler

```
package org.identityconnectors.flatfile.utils;
.
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.StringTokenizer;
.
import org.identityconnectors.flatfile.FlatFileConfiguration;
import org.identityconnectors.flatfile.FlatFileUserAccount;
import org.identityconnectors.flatfile.io.FlatFileMetadata;
.
/**
 * Class for the utility functions
 *
 * @author Admin
 *
 */
public class AccountConversionHandler {
.
    private FlatFileConfiguration fileConfig;
    private FlatFileMetadata metadata;
.
    /**
     * Instantiates the handler class. But needs the configuration
     *
     * @param metadata
     * @param fileConfig
     */
    public AccountConversionHandler(FlatFileMetadata metadata,
            FlatFileConfiguration fileConfig) {
        this.fileConfig = fileConfig;
        this.metadata = metadata;
    }
.
    /**
     * Converts strings records to the user account objects.
     *
     * @param accountRecord
     * @return
     * @throws RuntimeException
     *              If string is not formatted as per accepted standards
     */
   public FlatFileUserAccount convertStringRecordToAccountObj(
            String accountRecord) {
.
        StringTokenizer tokenizer = new StringTokenizer(accountRecord,
                fileConfig.getTextFieldDelimeter());
.
        // Assert number of columns matching with number of tokens
        if (metadata.isDifferentFromNumberOfFields(tokenizer.countTokens()))
            throw new RuntimeException(
                    "Number of tokens doesn't match number of columns");
.
        // Get the attributes
        List<String> attrNames = metadata.getOrderedTextFieldNames();
        Map<String, String> attrValMap = new HashMap<String, String>();
.
        // Number of tokens are same. Same loop will work
        for (String attrName : attrNames) {
            String attrVal = "";
            if (tokenizer.hasMoreTokens())
                attrVal = tokenizer.nextToken();
.
            attrValMap.put(attrName, attrVal);
        }
.
        // Assumption : All attributes are mandatory for user. Change with
the
        // change in assumption
        Set<String> mandatoryAttributeNames = attrValMap.keySet();
        FlatFileUserAccount userAccountRecordObj = new FlatFileUserAccount(
                mandatoryAttributeNames, attrValMap);
        return userAccountRecordObj;
.
   }
.
    /**
     * Converts account objects to storable string records
     *
     * @param accountObj
     * @return
     */
    public String convertAccountObjToStringRecord(
            FlatFileUserAccount accountObj) {
        StringBuilder strRecord = new StringBuilder();
.
        // Build the string record from the object
        List<String> attrNames = metadata.getOrderedTextFieldNames();

        int index=0;
        for (String attrName: attrNames) {
            String attrVal = accountObj.getAttributeValue(attrName);
            strRecord.append(attrVal);

           // Add delimeter
            if (index < attrNames.size()-1) {
                strRecord.append(fileConfig.getTextFieldDelimeter());
                index++;
            } else {
                // Record ended
                String newLineCharacter = "\n";
                strRecord.append(newLineCharacter);
                break;
            }
        }
        return strRecord.toString();
    }
.
    /**
     * Asserts if given object is not null
     *
     * @param message
     * @param obj
     */
    public void assertNotNull(String message, Object obj) {
        if (obj == null)
            throw new RuntimeException(message);
    }

}
```

Example 5–12 Messages.Properties

```
USER_ACCOUNT_STORE_HELP=File in which user account will be stored
USER_ACCOUNT_STORE_DISPLAY=User Account File
USER_STORE_TEXT_DELIM_HELP=Text delimeter used for separating the columns
USER_STORE_TEXT_DELIM_DISPLAY=Text Field Delimeter
UNIQUE_ATTR_HELP=The name of the attribute which will act as unique identifier
UNIQUE_ATTR_DISPLAY=Unique Field
CHANGELOG_ATTR_HELP=The name of the attribute which will act as changelog
CHANGELOG_ATTR_DISPLAY=Changelog Field
```

5.2 Uploading the Identity Connector Bundle to Oracle Identity Manager Database

The identity connector bundle must be available to ICF in Oracle Identity Manager database. Follow the list of sections in order to integrate the ICF identity connector with Oracle Identity Manager. Some of the procedures include configuration by using the Oracle Identity Manager Design Console.

- Registering the Connector Bundle with Oracle Identity Manager
- Creating Basic Identity Connector Metadata
- Creating Provisioning Metadata
- Creating Reconciliation Metadata

5.2.1 Registering the Connector Bundle with Oracle Identity Manager

The connector bundle must be available for the Connector Server local to Oracle Identity Manager. Following is the procedure to accomplish this:

1. Copy the connector bundle JAR to the machine on which Oracle Identity Manager in installed.
2. Run the following command to upload the JAR.

$*MW_HOME*/server/bin/UploadJars.sh

> **Note:** In this chapter, *DW_HOME* represents $*MW_HOME*/Oracle_IDM1.

3. Select ICFBundle as the JAR type.
4. Enter the location of the connector bundle JAR.
5. Press **Enter**.

5.2.2 Creating Basic Identity Connector Metadata

This metadata configuration is needed for both provisioning and reconciliation. The set of procedures in this section are completed by using the Oracle Identity Manager Design Console.

- Creating the IT Resource Type Definition
- Creating the Resource Object
- Creating the Main Configuration Lookup
- Creating Object Type Configuration Lookup

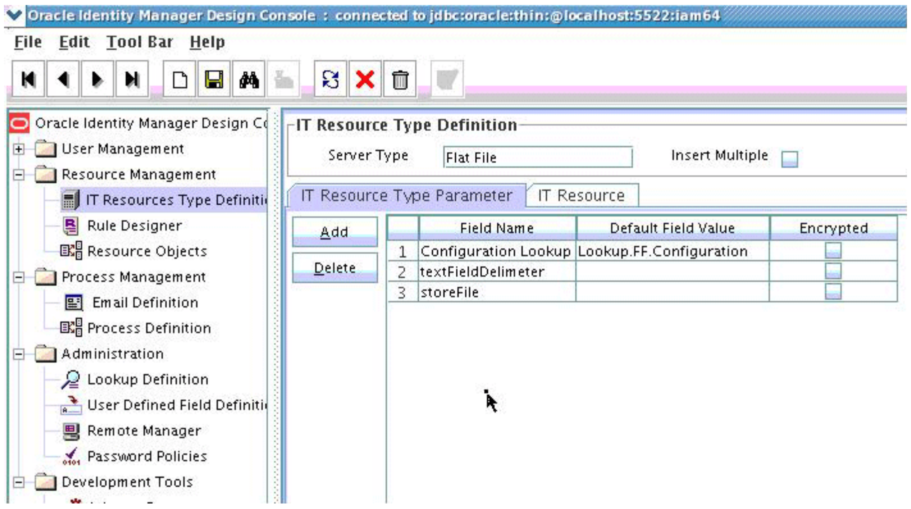
5.2.2.1 Creating the IT Resource Type Definition

An IT resource type definition is the representation of a resource's connection information. The configuration parameters in the IT resource type definition should be matched with the configuration parameters of the connector bundle. The values of the parameters in the IT resource will be set in the bundle configuration.

> **Note:** You may include parameters the bundle configuration is not using. They produce no negative effects on the bundle operations.

1. Log in to the Oracle Identity Manager Design Console.
2. Click IT Resource Type Definition under Resource Management.
3. Create a new IT Resource Type Definition with the Server Type defined as Flat File.
4. Add the following parameters as illustrated in Figure 5–1.
   - **Configuration Lookup** is the marker of the main configuration lookup for the resource. The name of the parameter must be Configuration Lookup. It is a good practice to add a value to Default Field Value.
   - **textFieldDelimeter** maps to the textFieldDelimeter parameter in the bundle configuration. The value of this parameter will be passed.
   - **storeFile** maps to the storeFile parameter in the bundle configuration. The value of this parameter will be passed.

Figure 5–1 IT Resource Type Definition in Design Console



Server Type: Flat File
Insert Multiple: ☐

	Field Name	Default Field Value	Encrypted
1	Configuration Lookup	Lookup.FF.Configuration	☐
2	textFieldDelimeter		☐
3	storeFile		☐

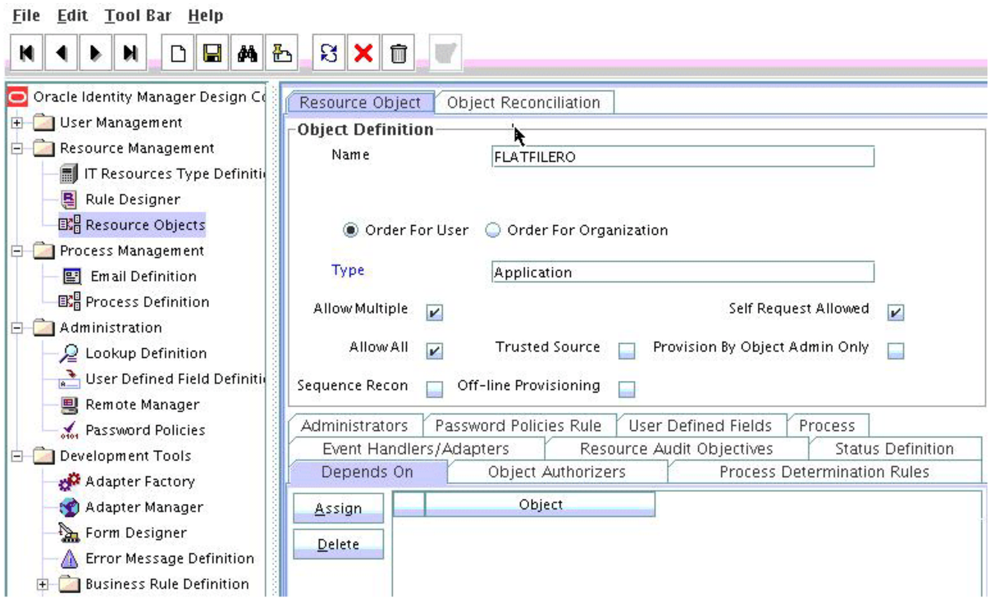
5.2.2.2 Creating the Resource Object

The resource object is the Oracle Identity Manager representation of a resource. The connector bundle is tied to the resource object.

1. Log in to the Oracle Identity Manager Design Console.
2. Click Resource Objects under Resource Management.
3. Create a new resource object with the name FLATFILERO.

   As the resource object is a target resource don't check the Trusted Source box as illustrated in Figure 5–2.

Figure 5–2 Resource Objects in Design Console



Name: FLATFILERO

[✓ Order For User] Order For Organization

Type: Application

Allow Multiple: ✓
Self Request Allowed: ✓
Allow All: ✓
Trusted Source: ☐
Provision By Object Admin Only: ☐
Sequence Recon: ☐
Off-line Provisioning: ☐

5.2.2.3 Creating Lookups

Separate lookups have to be defined for different objects supported by the connector bundle. This lookup can contain provisioning and reconciliation related information for those objects. The Main Configuration Lookup is the root for object specific lookups as it contains the pointers to those lookups. The following sections contain information on how to create lookups.

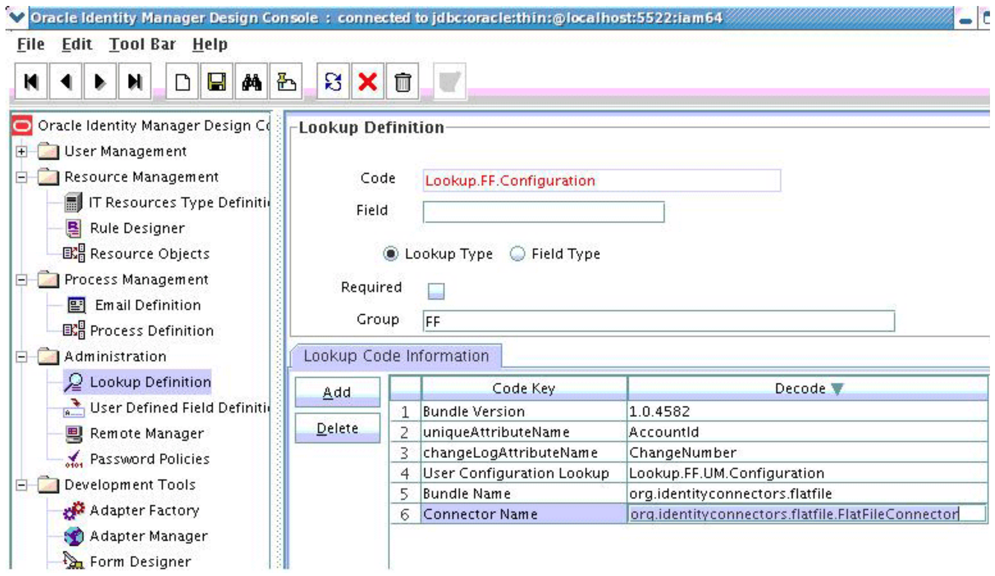
- Creating the Main Configuration Lookup
- Creating Object Type Configuration Lookup

5.2.2.3.1 Creating the Main Configuration Lookup The Configuration Lookup (as defined in Section 5.2.2.1, "Creating the IT Resource Type Definition") holds connector bundle configurations that are not counted as connection information. If a configuration parameter is not found in the IT Resource Type Definition, Oracle Identity Manager will look in the Configuration Lookup. The main Configuration Lookup contains bundle properties and bundle configurations. Bundle Property parameters are mandatory as they are needed for identifying the correct bundle. Bundle configurations that are not defined as part of the IT resource type definition (discussed in Section 5.2.2.1, "Creating the IT Resource Type Definition") can be declared here.

> **Note:** The values for Code Key should match exactly as illustrated. The values for Decode are specific to the connector bundle.

1. Log in to the Oracle Identity Manager Design Console.
2. Click Lookup Definition under Administration.
3. Create a new lookup and add Lookup.FF.Configuration as the value for Code.
4. Add the following Lookup Code Information as illustrated in Figure 5–3.
   - Add *VERSION* as the required Bundle Version.
   - Add **org.identityconnectors.flatfile** as the required Bundle Name.
   - Add **org.identityconnectors.flatfile.FlatFileConnector** as the required Connector Name.
   - Add AccountId as the value of uniqueAttributeName. AccountId is a unique string identifier that represents the account to be provisioned or reconciled. It is the name of the column in the flat file. AccountId is unique and is used to represent a user (account detail) uniquely.
   - Add ChangeNumber as the value of changeLogAttributeName. When an account is created, a number is attached to it indicating the total accounts created. This value is maintained in the variable called ChangeNumber.
   - *OBJECT_TYPE_NAME* Configuration Lookup is the configuration lookup for the particular object type. In this example, the object type is User as User Configuration Lookup is defined.

Figure 5–3   Lookup Definition in Design Console



Code: Lookup.FF.Configuration
Field:
[✓ Lookup Type] Field Type
Required: ☐
Group: FF

	Code Key	Decode
1	Bundle Version	1.0.4582
2	uniqueAttributeName	AccountId
3	changeLogAttributeName	ChangeNumber
4	User Configuration Lookup	Lookup.FF.UM.Configuration
5	Bundle Name	org.identityconnectors.flatfile
6	Connector Name	org.identityconnectors.flatfile.FlatFileConnector

5.2.2.3.2   Creating Object Type Configuration Lookup Object type configuration lookup contains the parameters specific to the particular object type. Object type is an entity over which an identity connector operates. It is mapped to ICF ObjectClass. In Section 5.2.2.3.1, "Creating the Main Configuration Lookup," User Configuration Lookup has been referenced so that User is the object type, in this case mapped to ObjectClass.ACCOUNT. (Roles and UserJobData are two other object types.) The object type name has to match with ObjectClass name supported by the identity connector bundle. The User object type is mapped to predefined ObjectClass.ACCOUNT, the Group object type is mapped to predefined ObjectClass.GROUP. If the identity connector supports multiple objects, then this step must be repeated for each.

Note: Because these use cases cover only the basic functionality, the configuration is kept to the mandatory attribute.

---

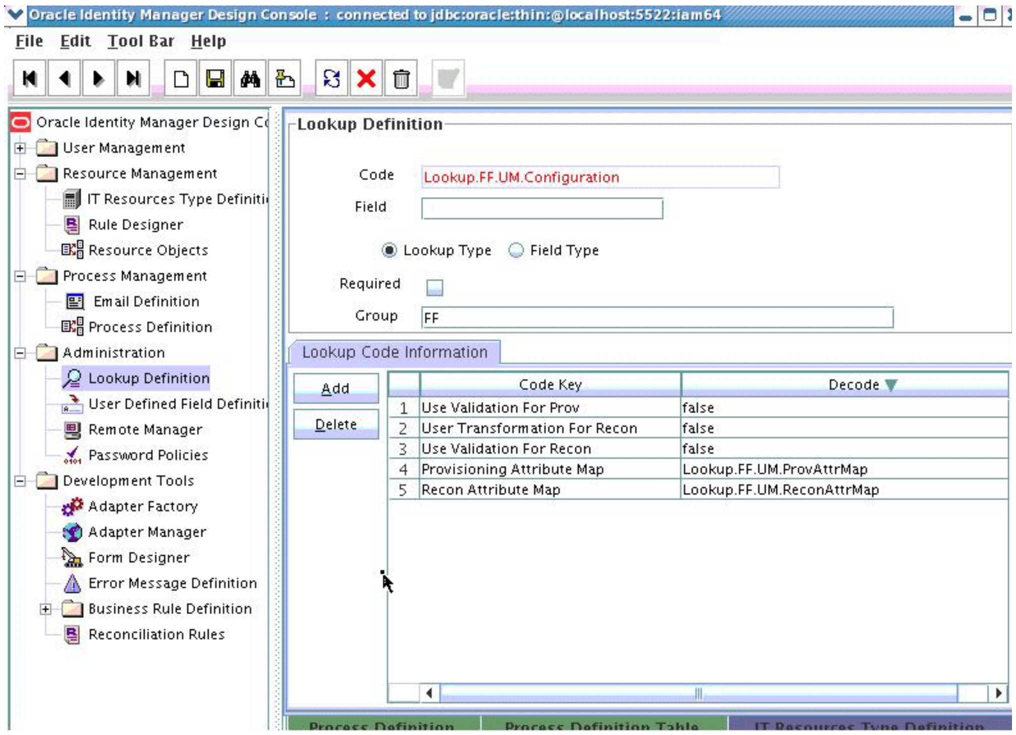
1. Log in to the Oracle Identity Manager Design Console.
2. Click Lookup Definition under Administration.
3. Create a new Lookup and add Lookup.FF.UM.Configuration as the Code.
4. Set the following attributes as illustrated in Figure 5–4.

Note: This tutorial focuses on the minimum configurations needed to run an identity connector.

- **Provisioning Attribute Map** takes a value of Lookup.FF.UM.ProvAttrMap. This lookup contains the mapping between Oracle Identity Manager fields and identity connector attributes. The mapping is used during provisioning.
- **Reconciliation Attribute Map** takes a value of Lookup.FF.UM.ReconAttributeMap. This lookup contains the mapping

between Oracle Identity Manager reconciliation fields and identity connector attributes. The mapping is used during reconciliation.

Figure 5–4 Second Lookup Definition in Design Console



Code: Lookup.FF.UM.Configuration

Field:

[✓ Lookup Type] Field Type

Required: ☐

Group: FF

	Code Key	Decode
1	Use Validation For Prov	false
2	User Transformation For Recon	false
3	Use Validation For Recon	false
4	Provisioning Attribute Map	Lookup.FF.UM.ProvAttrMap
5	Recon Attribute Map	Lookup.FF.UM.ReconAttrMap

5.2.3 Creating Provisioning Metadata

The following sections should be followed in order to configure Oracle Identity Manager for flat file provisioning.

- Creating a Process Form
- Creating Adapters
- Creating A Process Definition
- Creating a Provisioning Attribute Mapping Lookup

5.2.3.1 Creating a Process Form

A process form is used as the representation of object attributes on Oracle Identity Manager. This facilitates user input to set object attributes before passed to the connector bundle for an operation.

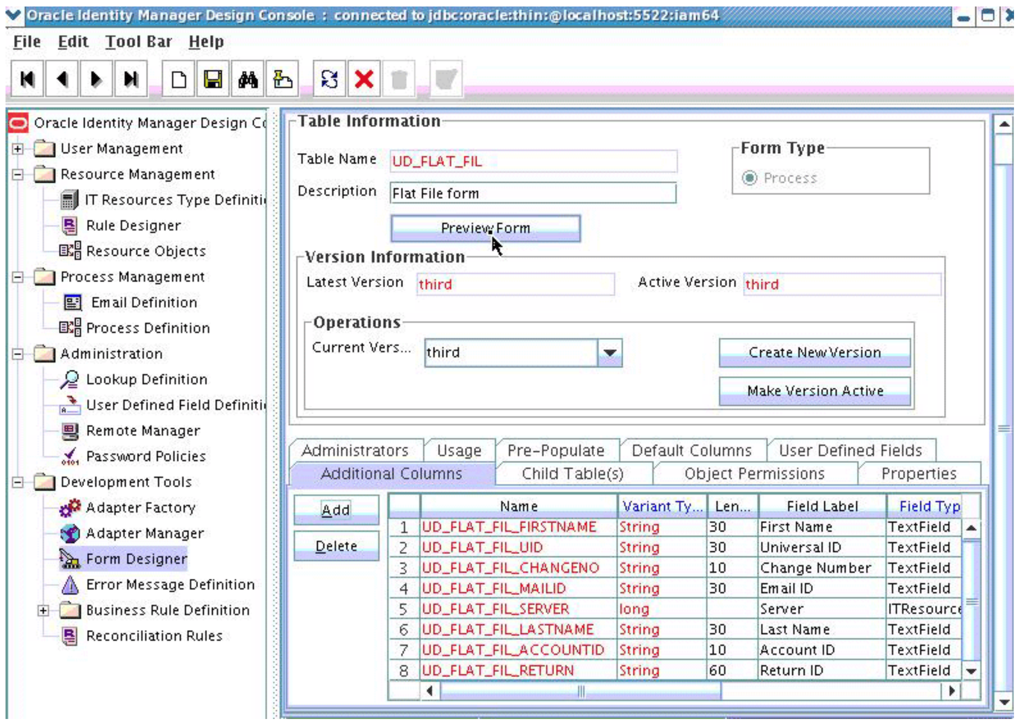
Attributes defined in the process form are not conventions. The form is a way to challenge the attributes that need to be passed to the identity connector. In general, define an attribute for each supported attribute in the identity connector.

Note: It is good practice to have a one to one mapping on the identity connector attributes.

There should be a field for querying the IT resource that should be associated with the respective IT Resource Type Definition. Variable type of each field should map to the type of the object attribute.

1. Log in to the Oracle Identity Manager Design Console.
2. Click Form Designer under Development Tools.
3. Create a new form with the Table Name UD_FLAT_FIL as illustrated in Figure 5–5.

Figure 5–5 Form Designer in Design Console


Oracle Identity Manager Design Console : connected to jdbc:oracle:thin:@localhost:5522:iam64

File Edit Tool Bar Help

Oracle Identity Manager Design Co
- User Management
- Resource Management
  - IT Resources Type Definiti
  - Rule Designer
  - Resource Objects
- Process Management
  - Email Definition
  - Process Definition
- Administration
  - Lookup Definition
  - User Defined Field Definiti
  - Remote Manager
  - Password Policies
- Development Tools
  - Adapter Factory
  - Adapter Manager
  - Form Designer
  - Error Message Definition
  - Business Rule Definition
  - Reconciliation Rules

Table Information

Table Name: UD_FLAT_FIL

Description: Flat File form

Preview Form

Form Type: Process

Version Information

Latest Version: third

Active Version: third

Operations

Current Vers...: third

Create New Version

Make Version Active

Administrators | Usage | Pre-Populate | Default Columns | User Defined Fields | Additional Columns | Child Table(s) | Object Permissions | Properties

Add

Delete

	Name	Variant Ty...	Len...	Field Label	Field Typ
1	UD_FLAT_FIL_FIRSTNAME	String	30	First Name	TextField
2	UD_FLAT_FIL_UID	String	30	Universal ID	TextField
3	UD_FLAT_FIL_CHANGENO	String	10	Change Number	TextField
4	UD_FLAT_FIL_MAILID	String	30	Email ID	TextField
5	UD_FLAT_FIL_SERVER	long		Server	ITResource
6	UD_FLAT_FIL_LASTNAME	String	30	Last Name	TextField
7	UD_FLAT_FIL_ACCOUNTID	String	10	Account ID	TextField
8	UD_FLAT_FIL_RETURN	String	60	Return ID	TextField

4. Add the attributes defined in the connector schema, as listed in Table 5–1.

Table 5–1 Form Designer Fields

Name	Variant	Field Label	Field Type
UD_FLAT_FIL_FIRSTNAME	String	First Name	TextField
UD_FLAT_FIL_UID	String	Universal ID	TextField
UD_FLAT_FIL_CHANGENO	String	Change Number	TextField
UD_FLAT_FIL_MAILID	String	Email ID	TextField
UD_FLAT_FIL_SERVER	long	Server	ITResource
UD_FLAT_FIL_LASTNAME	String	Last Name	TextField
UD_FLAT_FIL_ACCOUNTID	String	Account ID	TextField
UD_FLAT_FIL_RETURN	String	Return ID	TextField

Note: The flat file column names are FirstName, ChangeNo, EmailID, Server, LastName, and AccountID.

5. Click the Properties tab.
6. Add the following properties to Server(ITResourceLookupField) as illustrated in Figure 5–6.
   - Required = true
   - Type = Flat File

Figure 5–6 Properties of Form Designer in Design Console

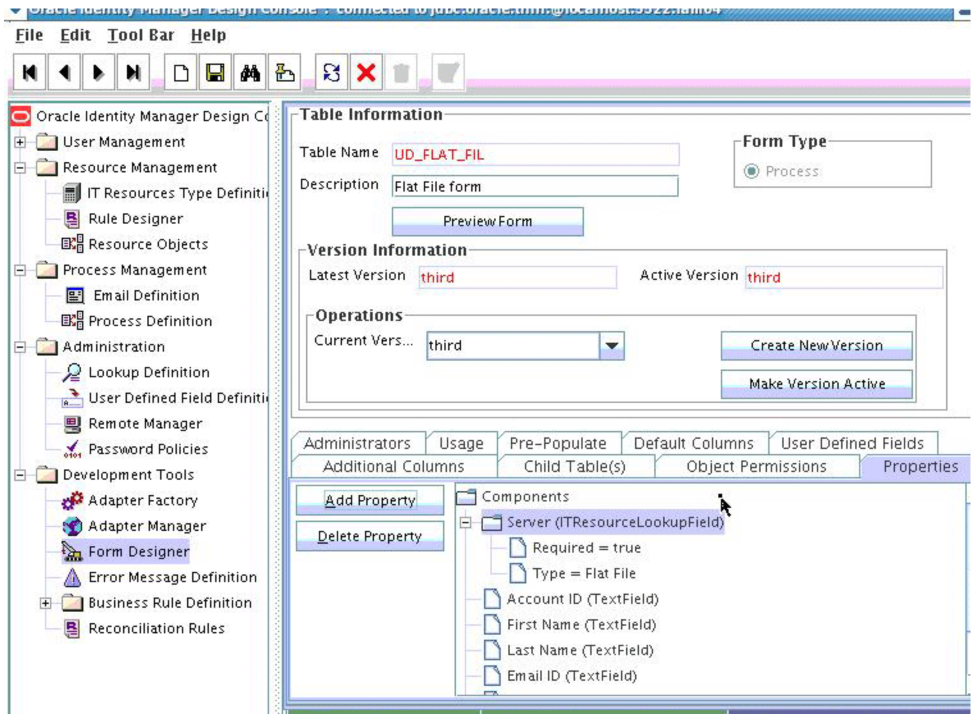


Table Information
Table Name: UD_FLAT_FIL
Description: Flat File form
Form Type: Process
Version Information
Latest Version: third
Active Version: third
Operations
Current Vers...: third

Components
Server (ITResourceLookupField)
Required = true
Type = Flat File
Account ID (TextField)
First Name (TextField)
Last Name (TextField)
Email ID (TextField)

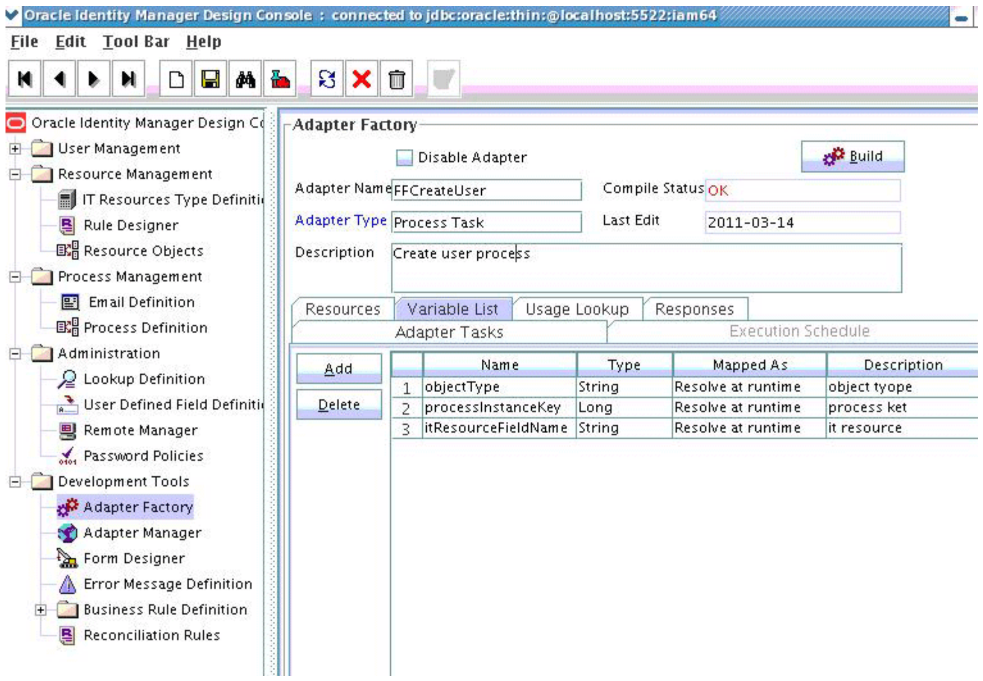
7. Save the form.
8. Click Make Version Active.

5.2.3.2 Creating Adapters

An adapter has to be created for all operations supported by the connector bundle, including Create, Update, and Delete.

1. Log in to the Oracle Identity Manager Design Console.
2. Click **Adapter Factory** under Development Tools.
3. Create a new adapter and add FFCreateUser as the Adapter Name.
4. Add Process Task as the Adapter Type.
5. Save the adapter.
6. Click the **Variable List** tab and add the following variables, as shown in Figure 5–7.
   - objectType with Type String and Mapped as Resolve at runtime.
   - processInstanceKey with Type long and Mapped as Resolve at runtime.
   - itResourceFieldName with Type String and Mapped as Resolve at runtime.

Figure 5–7 Adapter Factory Variable List in Design Console



Adapter Factory

Disable Adapter

Adapter Name: FFCreateUser

Compile Status: OK

Adapter Type: Process Task

Last Edit: 2011-03-14

Description: Create user process

	Name	Type	Mapped As	Description
1	objectType	String	Resolve at runtime	object tyope
2	processInstanceKey	Long	Resolve at runtime	process ket
3	itResourceFieldName	String	Resolve at runtime	it resource

7. Add a Java functional task to the adapter by following this sub procedure, as shown in Figure 5–8.

a. Click the Adapter Tasks tab.

b. Select the adapter and click Add.

c. Select Java from the task options.

d. Select **icf-oim-intg.jar** from the API source.

e. Select **oracle.iam.connetors.icfcommon.prov.ICProvisioninManager** as the API Source.

f. Select **createObject** as the method for the task.

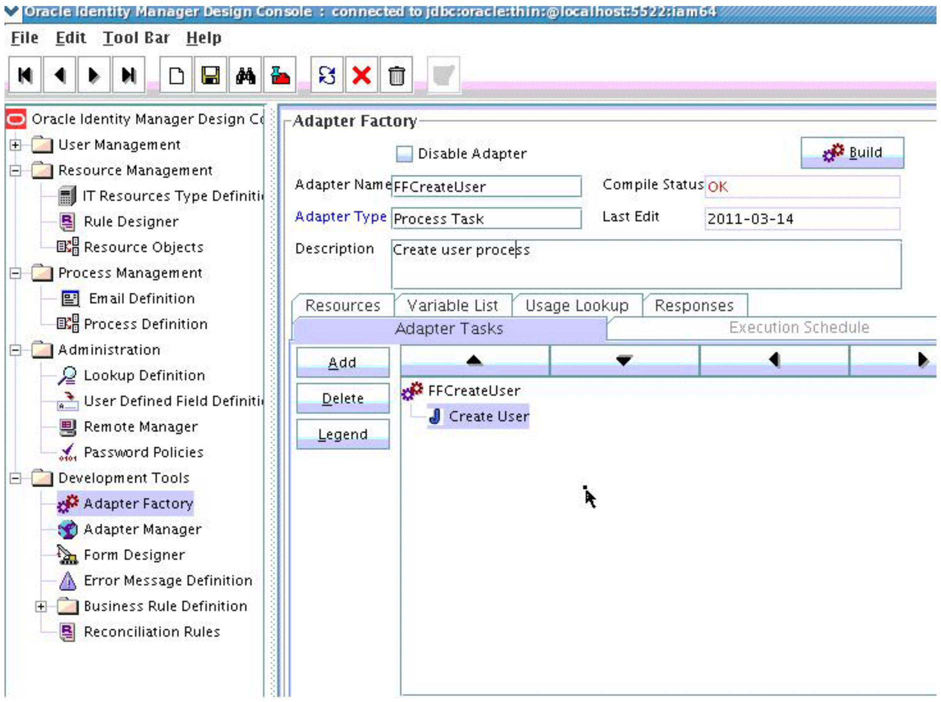
g. Save the configurations.

h. Map the variables (previously added to the Variables List) against the appropriate method inputs and outputs.

i. Map the configuration parameters against the appropriate method inputs and outputs.

Database Reference maps to Database Reference (Adapter References) and Return Variable maps to Return Variable (Adapter Variables).

Figure 5–8 Adapter Factory in Design Console



Adapter Name: FFCreateUser
Compile Status: OK
Adapter Type: Process Task
Last Edit: 2011-03-14
Description: Create user process

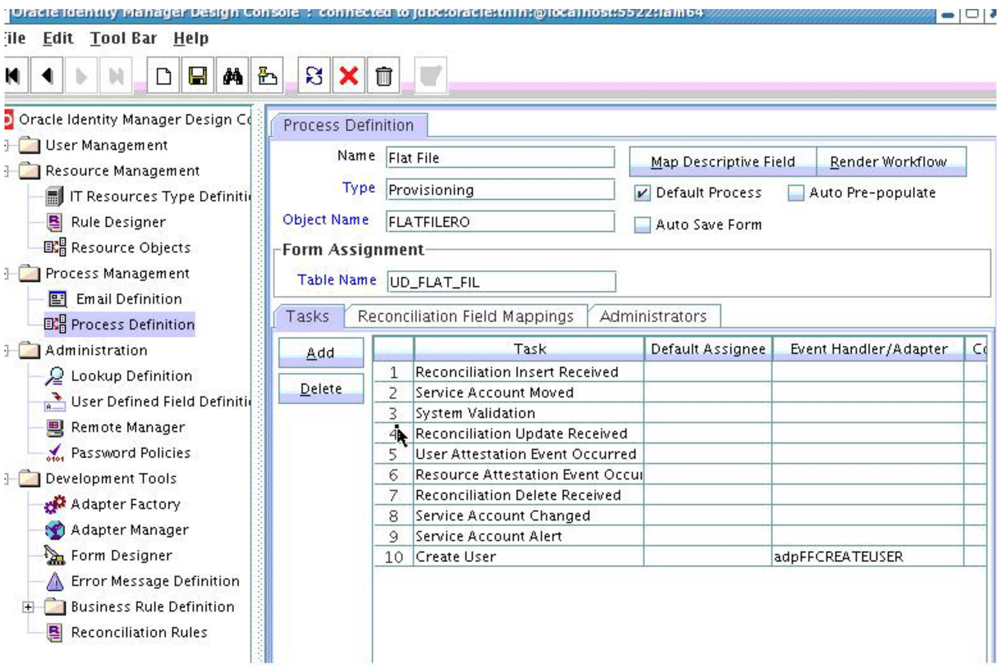
8. Save and build the adapter.

5.2.3.3 Creating A Process Definition

Process Definition defines the behavior of the connector bundle for a particular operation. Every operation has a corresponding task associated with it. This procedure will configure the process definition and integration of the process task for the Create operation.

1. Log in to the Oracle Identity Manager Design Console.
2. Click Process Definition under the Process Management tab.
3. Create a new process definition and name it Flat File as illustrated in Figure 5–9.

Figure 5–9 Process Definition in Design Console

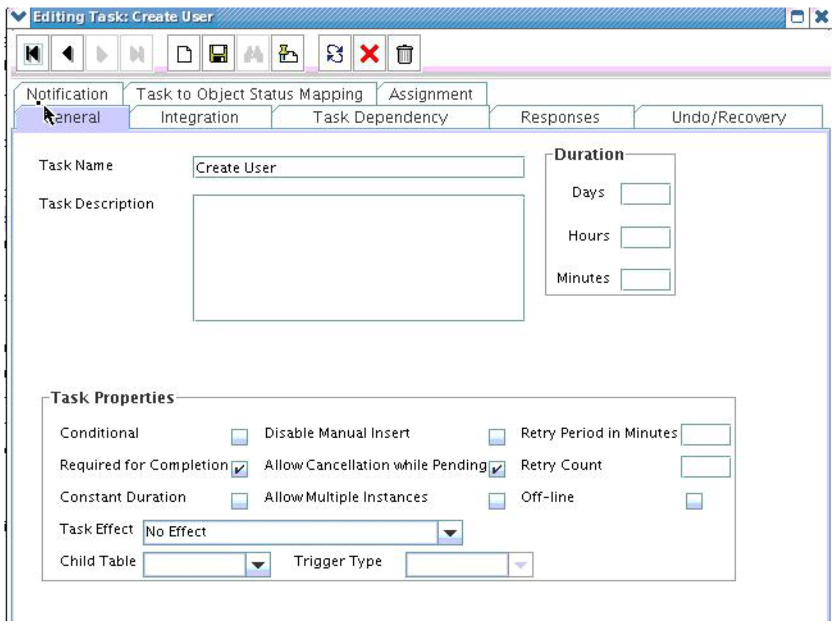


Name	Flat File
Type	Provisioning
Object Name	FLATFILERO
Default Process	✓
Auto Pre-populate	
Auto Save Form	
Table Name	UD_FLAT_FIL

	Task	Default Assignee	Event Handler/Adapter	Co
1	Reconciliation Insert Received			
2	Service Account Moved			
3	System Validation			
4	Reconciliation Update Received			
5	User Attestation Event Occurred			
6	Resource Attestation Event Occu			
7	Reconciliation Delete Received			
8	Service Account Changed			
9	Service Account Alert			
10	Create User		adpFFCREATEUSER	

4. Select Provisioning as the Type of process.
5. Provide the resource Object Name for the identity connector; in this example, FLATFILERO.
6. Provide the process form Table Name; in this example, UD_FLAT_FIL.
7. Add a process task and name it Create User.
8. Double click **Create User** to edit as illustrated in Figure 5–10.

Figure 5–10 Editing Task Screen in Design Console

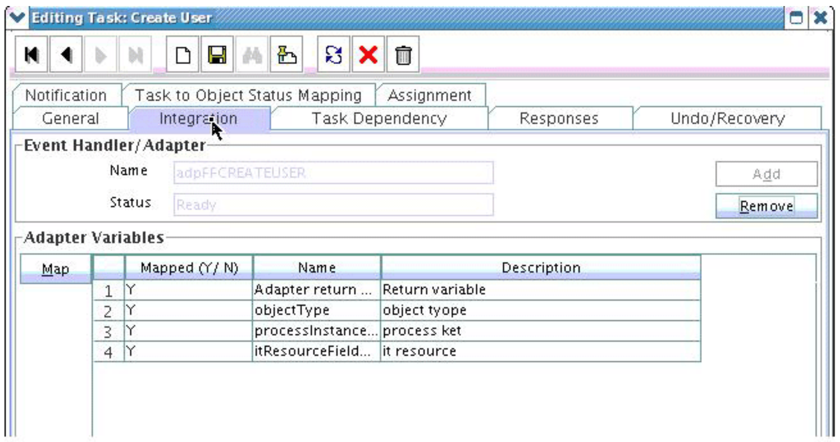


Task Name	Create User				
Task Description					
Conditional		Disable Manual Insert		Retry Period in Minutes	
Required for Completion	✓	Allow Cancellation while Pending	✓	Retry Count	
Constant Duration		Allow Multiple Instances		Off-line	
Task Effect	No Effect				
Child Table		Trigger Type			

9. Click the **Integration** tab.
10. Click Add and select the FFCreateUser adapter from the list as illustrated in Figure 5–11.

    The adapter will be available only after it is compiled.

Figure 5–11 Integration Tab in Design Console



Event Handler/Adapter

Name: adpFFCREATEUSER

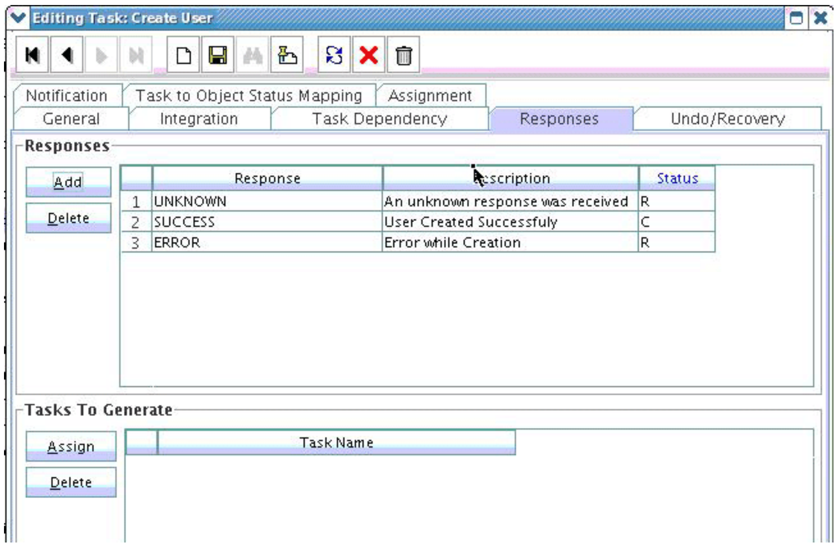
Status: Ready

Adapter Variables

	Mapped (Y/ N)	Name	Description
1	Y	Adapter return ...	Return variable
2	Y	objectType	object tyope
3	Y	processInstance...	process ket
4	Y	itResourceField...	it resource

11. Map the variables as follows to set the response code returned by the identity connector.
    - Adapter Return Variable – Response Code
    - Object Type – [Literal:String] User (Name of the object type)
    - Process Instance Key – [Process Data] Process Instance
    - IT Resource Field Name – [Literal:String] UD_FLAT_FIL_SERVER (Form field name that contains the IT resource information)
12. Click the Responses tab and configure the responses as illustrated in Figure 5–12.
    - UNKNOWN can be described as *Unknown response received* with a status of R (Rejected).
    - SUCCESS can be described as *Operation completed* with a status of C (Completed).
    - ERROR can be described as *Error occurred* with a status of R.

Figure 5–12 Configure Responses in Design Console


Editing Task: Create User

Notification | Task to Object Status Mapping | Assignment
General | Integration | Task Dependency | Responses | Undo/Recovery

Responses

Add
Delete

	Response	Description	Status
1	UNKNOWN	An unknown response was received	R
2	SUCCESS	User Created Successfuly	C
3	ERROR	Error while Creation	R

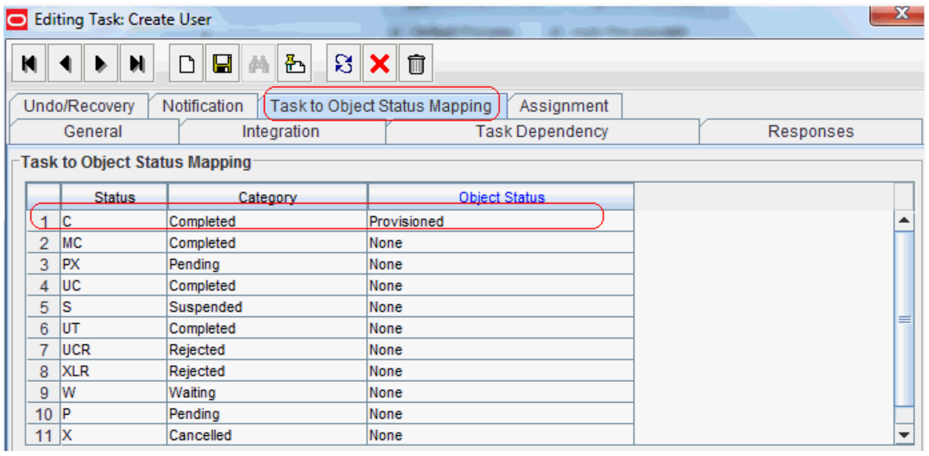
Tasks To Generate

Assign
Delete

	Task Name

13. Click the **Task to Object Status Mapping** tab.
14. Update the Object Status to **Provisioned** for Status C, as shown in Figure 5–13:

Figure 5–13 Task to Object Status Mapping


Editing Task: Create User

Undo/Recovery | Notification | Task to Object Status Mapping | Assignment
General | Integration | Task Dependency | Responses

Task to Object Status Mapping

	Status	Category	Object Status
1	C	Completed	Provisioned
2	MC	Completed	None
3	PX	Pending	None
4	UC	Completed	None
5	S	Suspended	None
6	UT	Completed	None
7	UCR	Rejected	None
8	XLR	Rejected	None
9	W	Waiting	None
10	P	Pending	None
11	X	Cancelled	None

15. Save the process task.

5.2.3.4 Creating a Provisioning Attribute Mapping Lookup

Provisioning Attribute Mapping Lookup contains mappings of Oracle Identity Manager fields to identity connector bundle attributes. In the Provisioning Attribute Mapping Lookup:

- Code keys are Field Labels of the process form.
- Decodes are identity connector bundle attributes.
- Child form attributes can be configured as embedded objects in inputs.
- The identity connector's provisioning operation returns the UID in response. This can be set in a form field by coding it against the identity connector bundle attribute.

Following is the procedure to create a Provisioning Attribute Mapping Lookup.

1. Log in to the Oracle Identity Manager Design Console.
2. Click Lookup Definition under the Administration tab.
3. Create a new lookup and name it Lookup.FF.UM.ProvAttrMap.

   The name of this lookup is referred from the object type configuration lookup. See Section 5.2.2.3.2, "Creating Object Type Configuration Lookup."
4. Add the form Field Labels as the code keys and identity connector bundle attributes as the decode.
   - Return ID : __UID__
   - Account ID: AccountId
   - Change Number: ChangeNumber
   - First Name: FirstName
   - Last Name: LastName
   - Email ID: MailId

5.2.3.4.1 Field Flags Used in the Provisioning Attributes Map

For provisioning attributes mapping, the following field flags can be appended to the code key:

- **LOOKUP:** This must be specified for all fields whose values are obtained by running a lookup reconciliation job. The values obtained from lookup reconciliation job have IT Resource Name/Key appended to it. Specifying this flag helps ICF integration to remove the appended value just before passing them onto the bundle. For example, the code key for a field with label Database whose value is obtained by running a lookup reconciliation job looks similar to Database[LOOKUP].

  Note: The LOOKUP flag can be specified for both Provisioning and Reconciliation Attribute Map. For provisioning, IT Resource Name/IT Resource Key prefix must be removed. For reconciliation, IT Resource Name/IT Resource Key prefix must be added.

- **IGNORE:** This must be specified for all fields whose values are to be ignored and not sent to bundle. For example, the code key for a field with label Database whose value need not be sent to bundle looks similar to Database[IGNORE].
- **WRITEBACK:** This must be specified for all fields whose values need to be written back into the process form right after the create or update operation. Adding this flag makes the ICF integration layer call ICF Get API to get values of attributes marked with the WRITEBACK flag. For example, the code key for a field with label Database whose value needs to be written back to the process form right after create/update looks similar to Database[WRITEBACK]. For this to work, the connector must implement the GetApiOp interface and provide an implementation for the ConnectorObject getObject(ObjectClass objClass,Uid uid,OperationOptions options) API. This API searches the target for the account whose Uid is equal to the passed in Uid, and builds a connector object containing all the attributes (and their values) that are to be written back to process form.

Note: If the connector does not implement the GetApiOp interface, then the WRITEBACK flag does not work and an error is generated.

- **DATE:** This must be specified for fields whose type need to be considered as Date, without which the values are considered as normal strings. For example, the code key for a field with label Today whose value needs to be displayed in the date format looks similar to Today[DATE].
- **PROVIDEONPSWDCHANGE:** This must be specified for all fields that need to be provided to the bundle(target) when a password update happens. Some targets expect additional attributes to be specified on every password change. Specifying the PROVIDEONPSWDCHANGE flag, tells ICF integration to send all the extra fields or attributes whenever a password change is requested. For example, the code key for a field with label Extra Attribute Needed for Password Change whose value needs to be provided to bundle(target) while password update looks similar to Extra Attribute Needed for Password Change[PROVIDEONPSWDCHANGE].

5.2.4 Creating Reconciliation Metadata

This section contains the procedures to configure the reconciliation of records from the flat file. You can use the target reconciliation as an example; trusted reconciliation can also be configured in a similar fashion. Do the procedures in the listed order.

1. Creating a Reconciliation Schedule Task
2. Creating a Reconciliation Profile
3. Setting a Reconciliation Action Rule
4. Creating Reconciliation Mapping
5. Defining a Reconciliation Matching Rule

5.2.4.1 Creating a Reconciliation Schedule Task

By default, reconciliation uses a Search operation on the connector bundle. This operation is invoked with a schedule task configured using Oracle Identity Manager. This procedure is comprised of the following sub procedures.

1. Defining the Schedule Task
2. Creating a Scheduled Task

5.2.4.1.1 Defining the Schedule Task To define the scheduled task:

1. Create a Deployment Manager XML file containing the scheduled task details as shown in Example 5–13. Make sure to update database value to your database.

Example 5–13 Deployment Manager XML with Scheduled Task Details

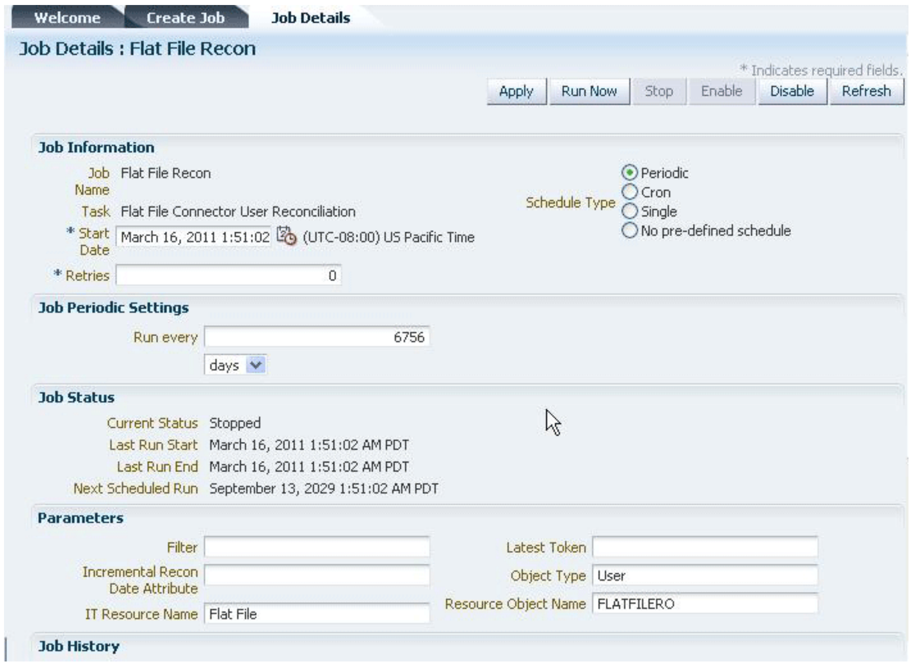
```
<?xml version = '1.0' encoding = 'UTF-8'?>
<xl-ddm-data version="2.0.1.0" user="XELSYSADM"
database="jdbc:oracle:thin:@localhost:5524/estView.regress.rdbms.dev.mycompany.com
" exported-date="1307546406635" description="FF">
<scheduledTask repo-type="MDS" name="Flat File Connector User Reconciliation"
mds-path="/db" mds-file="Flat File Connector User Reconciliation.xml">
    <completeXml>
        <scheduledTasks xmlns="http://xmlns.oracle.com/oim/scheduler">
            <task>
            <name>Flat File Connector User Reconciliation</name>
            <class>oracle.iam.connectors.icfcommon.recon.SearchReconTask</class>
            <description>Flat File Connector User Reconciliation</description>
            <retry>0</retry>
            <parameters>
              <string-param required="false" encrypted="false"
helpText="Filter">Filter</string-param>
              <string-param required="false" encrypted="false"
helpText="Incremental Recon Date Attribute">Incremental Recon Date
Attribute</string-param>
              <string-param required="false" encrypted="false" helpText="IT
Resource Name">IT Resource Name</string-param>
              <string-param required="false" encrypted="false" helpText="Object
Type">Object Type</string-param>
              <string-param required="false" encrypted="false" helpText="Latest
Token">Latest Token</string-param>
              <string-param required="false" encrypted="false" helpText="Resource
Object Name">Resource Object Name</string-param>
            </parameters>
          </task>
        </scheduledTasks>
    </completeXml>
</scheduledTask>
</xl-ddm-data>
```

2. Save the file as Flat File Connector User Reconciliation.xml.
3. Login to Oracle Identity System Administration. Under System Management, click **Import**.
4. Select the Flat File Connector User Reconciliation.xml file, and click **Import**.
5. Complete the steps in the wizard.

5.2.4.1.2 Creating a Scheduled Task This procedure explains how to create a scheduled task.

1. Log in to the Oracle Identity Manager Advanced Administration.
2. Click **Scheduler** under the System Management tab.
3. Add a schedule task and add Flat File Connector User Reconciliation as the type as illustrated in Figure 5–14.

Figure 5–14 The Scheduled Task Screen



Job Details : Flat File Recon

* Indicates required fields.

Job Information

Job Name: Flat File Recon

Task: Flat File Connector User Reconciliation

* Start Date: March 16, 2011 1:51:02 (UTC-08:00) US Pacific Time

* Retries: 0

Schedule Type: [✓ Periodic] Cron Single No pre-defined schedule

Job Periodic Settings

Run every: 6756 days

Job Status

Current Status: Stopped

Last Run Start: March 16, 2011 1:51:02 AM PDT

Last Run End: March 16, 2011 1:51:02 AM PDT

Next Scheduled Run: September 13, 2029 1:51:02 AM PDT

Parameters

Filter:

Incremental Recon Date Attribute:

IT Resource Name: Flat File

Latest Token:

Object Type: User

Resource Object Name: FLATFILERO

Job History

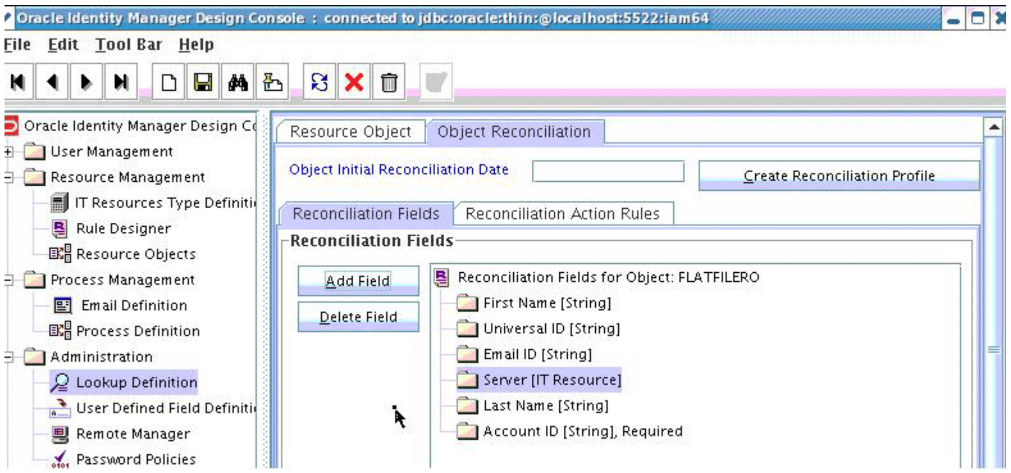
4. Set the parameters as follows:
   - IT Resource Name takes a value of Flat File.
   - Resource Object Name takes a value of FLATFILERO.
   - Object Type takes a value of User.
5. Click **Apply**.

5.2.4.2 Creating a Reconciliation Profile

A reconciliation profile defines the structure of the object attributes while reconciliation. The reconciliation profile should contain all the attributes that have reconciliation support.

1. Log in to the Oracle Identity Manager Design Console.
2. Click **Resource Objects** under Resource Management.
3. Open the FLATFILERO resource object.
4. Click the **Object Reconciliation** tab as illustrated in Figure 5–15.

Figure 5–15 Object Reconciliation in Design Console



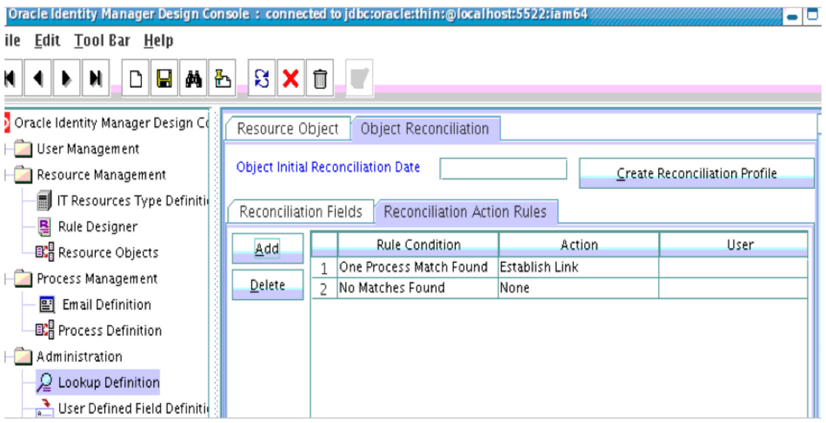
5. Add following reconciliation fields:
   - First Name [String]
   - Universal ID [String]
   - Email ID [String]
   - IT Resource Name [String]
   - Last Name [String]
   - Account ID [String], Required
6. Save the configuration.

5.2.4.3 Setting a Reconciliation Action Rule

A Reconciliation Action Rule defines the behavior of reconciliation. In this procedure, define the expected action when a match is found. This procedure assumes you are logged into the Oracle Identity Manager Design Console.

1. Open the FLATFILERO resource object.
2. Click the **Object Reconciliation** tab.
3. Click the **Reconciliation Action Rules** tab in the right frame as illustrated in Figure 5–16.

Figure 5–16 Reconciliation Action Rules in Design Console


Oracle Identity Manager Design Console : connected to jdbc:oracle:thin:@localhost:5522:iam64
ile Edit Tool Bar Help
Oracle Identity Manager Design Co
User Management
Resource Management
IT Resources Type Definiti
Rule Designer
Resource Objects
Process Management
Email Definition
Process Definition
Administration
Lookup Definition
User Defined Field Definiti
Resource Object
Object Reconciliation
Object Initial Reconciliation Date
Create Reconciliation Profile
Reconciliation Fields
Reconciliation Action Rules
Add
Delete

	Rule Condition	Action	User
1	One Process Match Found	Establish Link	
2	No Matches Found	None	

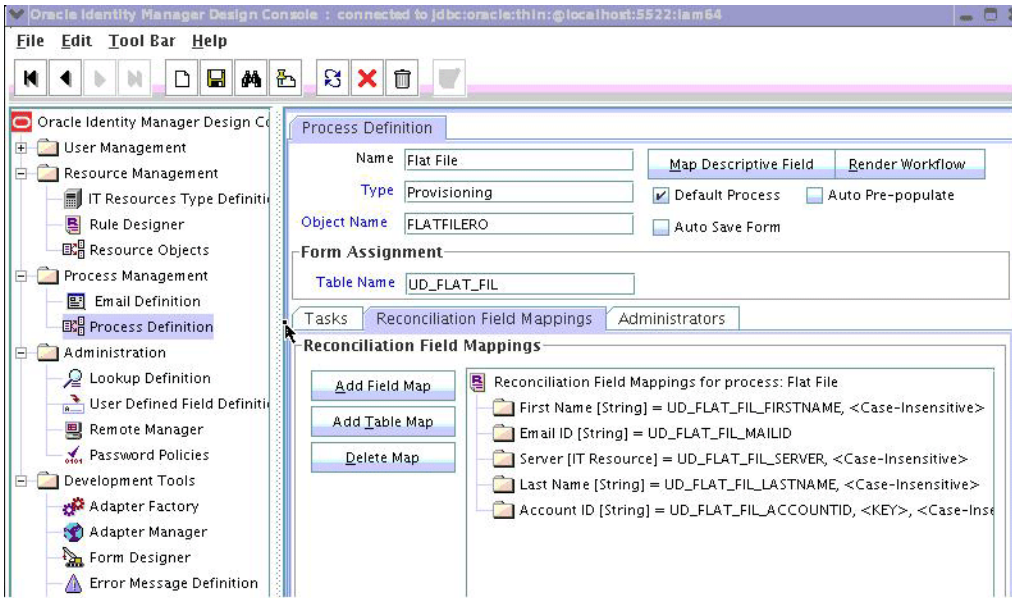
4. Add an action rule defined as One Process Match Found (Rule Condition) and Establish Link (Action).
5. Add an action rule defined as One Entity Match Found (Rule Condition) and Establish Link (Action).
6. Click **Create Reconciliation Profile**.
7. Click **Save**.

5.2.4.4 Creating Reconciliation Mapping

The reconciliation mapping has to be done in the process definition. This is to map the supported reconciliation fields (from resource object) to the process form fields. This mapping is needed only for configuring target reconciliation.

1. Log in to the Oracle Identity Manager Design Console.
2. Click Process Definition under Process Management.
3. Open the Flat File process definition.
4. Click the Reconciliation Field Mappings tab as illustrated in Figure 5–17.

Figure 5–17 Reconciliation Field Mapping in Design Console



Name: Flat File
Type: Provisioning
Object Name: FLATFILERO
☑ Default Process ☐ Auto Pre-populate
☐ Auto Save Form

Form Assignment
Table Name: UD_FLAT_FIL

Reconciliation Field Mappings

Reconciliation Field Mappings for process: Flat File
- First Name [String] = UD_FLAT_FIL_FIRSTNAME, <Case-Insensitive>
- Email ID [String] = UD_FLAT_FIL_MAILID
- Server [IT Resource] = UD_FLAT_FIL_SERVER, <Case-Insensitive>
- Last Name [String] = UD_FLAT_FIL_LASTNAME, <Case-Insensitive>
- Account ID [String] = UD_FLAT_FIL_ACCOUNTID, <KEY>, <Case-Inse

5. Add mappings between the reconciliation profile fields and the process form fields.

   - First Name[String] = UD_FLAT_FIL_FIRSTNAME
   - Email ID[String] = UD_FLAT_FIL_MAILID
   - IT Resource Name[String] = UD_FLAT_FIL_SERVER
   - Last Name[String] = UD_FLAT_FIL_LASTNAME
   - Account ID [String] = UD_FLAT_FIL_ACCOUNTID <KEY>

     <KEY> sets Account ID as a key field.

6. Save the configuration.

5.2.4.4.1 Field Flags Used in the Reconciliation Attributes Map

For reconciliation attributes mapping, the following field flags can be appended to the code key:

- **TRUSTED:** This must be specified in the Recon Attribute Map for the field that represents the status of the account. This flag must be specified only for trusted reconciliation. If this is specified, then the status of the account is either Active or Disabled. Otherwise, the status is either Enabled or Disabled. For example, the code key for a field with label Status whose value needs to be either Active/Disabled must look similar to Status[TRUSTED].

- **DATE:** In Recon Attribute Map, this must be specified for fields whose type need to be considered as Date. For example, the code key for a field with label Today whose value needs to be displayed in the date format must look similar to Today[DATE].

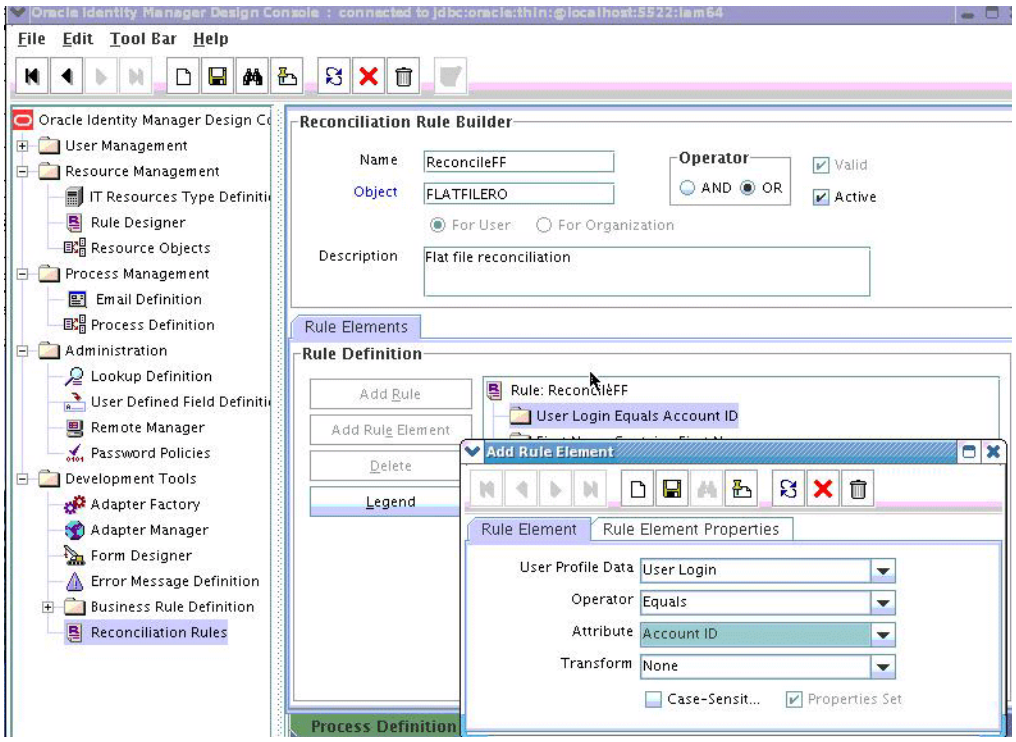
5.2.4.5 Defining a Reconciliation Matching Rule

A reconciliation matching rule defines the equation for calculating the user match.

1. Log in to the Oracle Identity Manager Design Console.

2. Open the Reconciliation Rules form under Development Tools.
3. Click **Add Rule**.

Figure 5–18 Adding Reconciliation Matching Rule



Name: ReconcileFF
Object: FLATFILERO
Operator: AND [✓ OR]
[✓ Valid] [✓ Active]
[✓ For User] For Organization
Description: Flat file reconciliation

Rule: ReconcileFF
User Login Equals Account ID

User Profile Data: User Login
Operator: Equals
Attribute: Account ID
Transform: None
Case-Sensit... [✓ Properties Set]

4. Select resource object FLATFILERO.
5. Save and add the rule element.

   User Login from the user profile data equals the Account ID resource attribute.
6. Save the rule.

5.3 Provisioning a Flat File Account

The flat file connector is ready to work. Now, the user needs to log in to Oracle Identity Manager and create an IT resource (target) using the following procedure.

- Create IT resource of type "Flat File".
- Provide the IT resource parameters as appropriate.
- Provide the configuration parameters in Lookup.FF.Configuration as appropriate.

5.4 Installing the Java Connector Server

Note: Connector Server version 12.2.1.3.0 is backward compatible with earlier versions of the Connector Server and can be used for all existing ICF Connectors.

To install the Java Connector Server:

1. Download the Connector Server package (`Connector_Server_122130_java.zip`) from the Oracle Technology Network site at the following URL:

   `http://www.oracle.com/technetwork/index.html`

2. Extract the contents of the Connector Server package and locate the `connector_server_java-1.5.0.zip` file.
3. Create a directory where you want to install Java Connector Server. This will be `CONNECTOR_SERVER_HOME`.
4. Extract the contents of the `connector_server_java-1.5.0.zip` file to `CONNECTOR_SERVER_HOME` directory.
5. In the `CONNECTOR_SERVER_HOME/conf/ConnectorServer.properties` file, set the properties as required by your deployment.

   The following example snippet shows the ConnectorServer.properties shipped with Java Connector Server:

```
.
##
## The port we are to run on
##
connectorserver.port=8759
##
## The bundle directory in which to find the bundles
##
connectorserver.bundleDir=bundles
.
##
## The bundle directory in which to find any libraries needed by bundles at
runtime
##
connectorserver.libDir=lib
.
##
## Set to true to use SSL.
## NOTE: Check also the following settings which are related to SSL:
## connectorserver.promptKeyStorePassword
## connectorserver.keyStore
## connectorserver.keyStoreType
## connectorserver.keyStorePassword
connectorserver.usessl=false
##
## Protocol in use for SSL communication e.g. TLSv1, TLSv1.1, TLSv1.2
##
connectorserver.protocol=TLSv1
.
##
## If set to true the user is prompted for key store password at startup.
## If set to false the key store password needs to be set with
-setKeyStorePassword command first.
##
connectorserver.promptKeyStorePassword=true
.
##
## Full path to key store.
##
connectorserver.keyStore=/tmp/KeyStore.jks
.
```

```
##
## KeyStore type
##
#connectorserver.keyStoreType=JKS
.
##
## Encrypted password. Set this by using the -setKeyStorePassword flag.
## It is used only if connectorserver.promptKeyStorePassword is set to false.
##
connectorserver.keyStorePassword=
.
##
## Optionally specify a specific address to bind to
##
#connectorserver.ifaddress=localhost
.
##
## Secure hash of the gateway key. Set this by using the
## -setKey flag
##
connectorserver.key=lmA6bMfENJGlIDbfrVtklXFK32s\=
.
##
## Use standard JDK logging
##
connectorserver.loggerClass=org.identityconnectors.common.logging.impl.JDKLogge
r
```

6. The `CONNECTOR_SERVER_HOME/conf` directory also contains the `logging.properties` file, which you can edit if required by your deployment.

> **Note:** The `logging.properties` file allows you to enable or disable logging and update the level information for log files. By default, logging is enabled and level is set to INFO.

5.5 Configuring the Java Connector Server without SSL for Oracle Identity Manager

To configure the Java Connector Server without SSL:

1. In the `$CONNECTOR_SERVER_HOME/conf/ConnectorServer.properties` file, set the `connectorserver.key` property by running the Java Connector Server with the `/setKey` option.

   For Java Connector Server on Windows, go to `$CONNECTOR_SERVER_HOME\bin` directory and find the ConnectorServer.bat script. Run the script:

   ```
   ./ ConnectorServer.bat /setKey <KEY>
   ```

   For Java Connector Server on Solaris and Linux, go to `$CONNECTOR_SERVER_HOME\bin` directory and find the `ConnectorServer.sh` script. Run the script:

   ```
   ./ ConnectorServer.sh /setKey <KEY>
   ```

2. For all other properties, edit the `ConnectorServer.properties` file manually.

   See Installing the Java Connector Server for an example snippet of the ConnectorServer.properties shipped with Java Connector Server.

5.6 Configuring the Java Connector Server with SSL for Oracle Identity Manager

You can configure SSL for Java Connector Server by providing the key store credentials in the `ConnectorServer.properties` file.

To do so:

1. Create a keystore that will be used for SSL communication between Oracle Identity Manager and Java connector server. To do so:

   a. On the host on which Java connector server is installed, locate the JAVA home directory.

   b. From the JAVA home directory, run the following command to generate a keystore:

   ```
   $JAVA_HOME/jre/bin/keytool -genkey {-alias ALIAS} {-keyalg KEYALG}
   {-keysize KEYSIZE} {-sigalg SIGALG} [-dname DNAME] [-keypass KEYPASS]
   {-validity VAL_DAYS} {-storetype STORETYPE} {-keystore KEYSTORE}
   [-storepass STOREPASS]
   ```

   For example:

   ```
   /scratch/jdk1.7.0_131/jre/bin/keytool -genkey
   -alias javaconnectorserver
   -keyalg RSA
   -keysize 2048
   -sigalg SHA256withRSA
   -dname "CN=localhost, OU=Identity, O=Oracle Corporation,C=US"
   -keypass weblogic1
   -keystore javaconnectorserver.jks
   -storepass weblogic1
   ```

   c. Export the certificate of the newly generated keystore to a file by running the following keytool command:

   ```
   $JAVA_HOME/jre/bin/keytool -export {-alias ALIAS} {-file CERT_FILE}
   {-storetype STORETYPE} {-keystore KEYSTORE} [-storepass STOREPASS]
   ```

   For example:

   ```
   $JAVA_HOME/jre/bin/keytool -export -alias javaconnectorserver
     -file javaconnectorserver.cert
     -keypass weblogic1
     -keystore javaconnectorserver.jks
     -storepass weblogic1
   ```

   d. Copy the certificate of Java connector server keystore on the Oracle Identity Manager host. Import this certificate of Java connector server keystore into the trust store used in Oracle Identity Manager by running the following command:

   ```
   $JAVA_HOME/jre/bin/keytool -import {-alias ALIAS} {-file CERT_FILE}
   [-keypass KEYPASS] {-noprompt} {-trustcacerts} {-storetype STORETYPE}
   {-keystore KEYSTORE} [-storepass STOREPASS]
   ```

   If Oracle Identity Manager is using custom identity and custom trust, then import the following certificate in custom trust and Java standard trust.

   ```
   $JAVA_HOME/jre/bin/keytool -import -alias javaconnectorservertrust
   -trustcacerts -file /scratch/javaconnectorserver.cert -keystore
   ```

```
DOMAIN_HOME/config/fmwconfig/CUSTOM_TRUST_KEYSTORE -storepass weblogic1
```

If Oracle Identity Manager is using custom identity and Java standard trust, then import this certificate in Java standard trust.

```
$JAVA_HOME/jre/bin/keytool -import -alias javaconnectorservertrust
-trustcacerts -file /scratch/javaconnectorserver.cert -keystore
JAVA_HOME/jre/lib/security/cacerts -storepass changeit
```

If Oracle Identity Manager is using Demo Identity and Demo Trust, then import the following certificate in `DOMAIN_HOME/config/fmwconfig/default-keystore.jks` file of Oracle Identity Manager and in Java standard trust.

```
$JAVA_HOME/jre/bin/keytool -import -alias javaconnectorservertrust
-trustcacerts -file /scratch/javaconnectorserver.cert -keystore
DOMAIN_HOME/config/fmwconfig/default-keystore.jks -storepass weblogic1
```

2. Provide the location of this Java connector server keystore in the $*CONNECTOR_SERVER_HOME*/conf/ConnectorServer.properties file:

   ```
   connectorserver.usessl=true
   connectorserver.keyStore={full path to your keystore file}
   connectorserver.keyStoreType=JKS (optionally you can set key store type, if
   not set JSK is used by default)
   ```

3. Provide the password of this Java Connector Server keystore in the `$CONNECTOR_SERVER_HOME`/conf/ConnectorServer.properties file. You can do the following:

   a. Set `connectorserver.promptKeyStorePassword=false` in `ConnectorServer.properties` and set the password as:

      ```
      cd $CONNECTOR_SERVER/bin
      ```

      For UNIX:

      ```
      connectorserver.sh /setKeyStorePassword thepassword
      ```

      For Windows:

      ```
      ConnectorServer.bat /setKeyStorePassword thepassword
      ```

      This command will set the encrypted value to `connectorserver.keyStorePassword` in `ConnectorServer.properties`.

      or

   b. Prompt to enter the keystore password every time you start the connector server by setting `connectorserver.promptKeyStorePassword=true` in `ConnectorServer.properties` file.

4. You can set the protocol for secure communication by setting the `connectorserver.protocol property` in `$CONNECTOR_SERVER_HOME/conf/ConnectorServer.properties` file as:

   ```
   ## Protocol in use for SSL communication e.g. TLSv1, TLSv1.1, TLSv1.2
   ##
   connectorserver.protocol=TLSv1
   ```
   Default value for this property is `TLSv1` for TLS 1.0 protocol.

Note: You can configure SSL between Java Connector Server and Target System. To do so:

- Check for the JAVA_HOME folder path in the Java Connector Server machine.
- Import target system certificate in Java standard trust store (<JAVA_HOME>/jre/lib/security/cacerts) of Java Connector Server machine using below command:

  ```
  keytool -import -alias oidstore -keystore
  JAVA_HOME/jre/lib/security/cacerts -file
  /scratch/cert/b64certificate.txt -storepass mypassword
  ```

  Where, oidstore is the alias, JAVA_HOME is the java home folder in the Java Connector Server machine, /scratch/cert/b64certificate.txt is the target system certification file, and mypassword is the password.

5.7 Upgrading the Java Connector Server

In the 12.2.1.3.0 version of the Connector Server pack, you can select the protocol for SSL communication between Oracle Identity Manager and Java Connector Server by using the `connectorserver.protocol` property. The supported values of this property are `TLSv1`, `TLSv1.1`, and `TLSv1.2`. Here, `TLSv1` denotes TLS 1.0 protocol, `TLSv1.1` denotes TLS 1.1 protocol, and `TLSv1.2` denotes TLS 1.2 protocol. The default value of this system property is `TLSv1`, which denotes TLS 1.0 protocol.

To upgrade Java Connector Server:

1. Stop the connector server service.
2. Create a backup of the directory on which Connector Server is installed.
3. Download the Connector Server package (`Connector_Server_122130_java.zip`) from the Oracle Technology Network site at the following URL:

   http://www.oracle.com/technetwork/index.html
4. Extract the contents of the Connector Server package (`Connector_Server_122130_java.zip`) and locate the `connector_server_java-1.5.0.zip` file.
5. Extract the contents of the `connector_server_java-1.5.0.zip` file in a directory.
6. Copy the files in the `connector_server_java-1.5.0/bin/` and `connector_server_java-1.5.0/lib/` directories from the 12.2.1.3.0 Java Connector Server pack to the installed location of Java Connector Server.
7. Open `connector_server_java-1.5.0/conf/ConnectorServer.properties` file from 12.2.1.3.0 Java connector server pack and open `conf/ConnectorServer.properties` file from installed location of Java Connector Server.
8. Add the following section of the `conf/ConnectorServer.properties` file at installed location from `connector_server_java-1.5.0/conf/ConnectorServer.properties` file in 12.2.1.3.0 Java connector server pack:

   ```
   ##
   ## Protocol in use for SSL communication e.g. TLSv1, TLSv1.1, TLSv1.2
   ```

```
connectorserver.protocol=TLSv1
```

This property provides an option to select the protocol for SSL communication. By default, the value is TLS1.0.

Note: Customizations are preserved during the upgrade of the connector server. If you have any customization in any of the updated files, then redo the same customizations form the backed up file.

---

9. Start the connector server after doing all the required settings.

6 Developing Identity Connectors Using .NET

This chapter is a tutorial that walks through the procedures necessary to develop an identity connector in .NET using the Identity Connector Framework (ICF) and the Oracle Identity Manager metadata. It includes information about important ICF classes and interfaces, the connector bundle, the connector server, and code samples for implementing a flat file .NET identity connector and creating Oracle Identity Manager metadata for user provisioning and reconciliation processes. It contains the following sections:

- Section 6.1, "Developing a Flat File .NET Connector"
- Section 6.2, "Deploying the Identity Connector Bundle on .NET Connector Server"
- Section 6.3, "Provisioning a Flat File Account"

6.1 Developing a Flat File .NET Connector

The procedure for developing a flat file connector is to develop an implementation of the Configuration interface followed by the implementation of the Connector class. The document discusses sample implementation of a flat file connector showing Create, Delete, Update and Search operations. To keep implementations and documentation simple, Configuration properties and Schema supported by connector have been kept to a minimum. This connector implementation should only be used as a sample which would help to create actual connectors.

To keep the connector implementation simple, lets assume that flat file has only Name, Gender, Qualification, Age attributes. You have only two configurations File Location and Delimiter. Rest configurations would be hardcoded in the sample.

1. Setting up the project in Microsoft Visual Studio and using the connector:

   a. Create a new visual studio project of type library.

   b. Make sure to add the following dlls as references:

      - Common.dll

      - Framework.dll

      - FrameworkInternal.dll

      - System.dll

      - System.Core.dll

      These dlls should be available with the .NET connector server.

2. Implement the configuration class for the Flat File Connector by extending the Org.IdentityConnectors.Framework.Spi.AbstractConfiguration base class.

Example 6–1 Implementation of AbstractConfiguration

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Org.IdentityConnectors.Framework.Spi;?
using Org.IdentityConnectors.Framework.Common.Exceptions;?
using System.IO;

namespace Org.IdentityConnector.FlatFileConnector?
{
    /// <summary>
    /// Configuration class for flat file connector representing target system
information?
    /// </summary>
    public class FlatFileConfiguration : AbstractConfiguration?
    {
        #region FileName
        /// <summary>
        /// Target file name
        /// </summary>
        /// <value>
        /// File name with complete path. As for executing the .NET Connector
bundle we need .NET Connector Server, hence the file should reside
        /// on the machine where the connector server is present.
        /// </value>
        [ConfigurationProperty(Required = true, Order = 1)]
        public String FileName { get; set; }
        #endregion

        #region Delimiter
        /// <summary>
        /// Delimiter used within the target flat file
        /// </summary>
        /// <value>
        /// Delimter
        /// </value>
        [ConfigurationProperty(Required = true, Order = 2)]
        public String Delimiter { get; set; }
        #endregion

        #region
        /// <summary>
        /// Validates if the configuration properties provided are as requiered,
if not throw ConfigurationException
        /// </summary>
        public override void Validate()
        {
            if (this.FileName == null || this.FileName.Length == 0)
            {
                throw new ConfigurationException("Configuration property FileName
cannot be null or empty");
            }
            if (!File.Exists(this.FileName))
            {
                throw new ConfigurationException("Target file " + this.FileName +
" does not exist");
            }
            if (this.Delimiter == null || this.Delimiter.Length == 0)
            {
                throw new ConfigurationException("Configuration property Delimiter
cannot be null or empty");
            }
        }
        #endregion
    }
}
```

3. Create connector class for the Flat File Connector by implementing different SPI interfaces `Org.IdentityConnectors.Framework.Spi`.

   Example 6–2 implements the PoolableConnector,CreateOp,SchemaOp,TestOp,DeleteOp,UpdateOp,SearchOp<String> interfaces and thus supports all CRUD operations

Example 6–2 Implementation of PoolableConnector

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Org.IdentityConnectors.Framework.Spi;
using System.IO;
using Org.IdentityConnectors.Framework.Common.Exceptions;
using System.Security.AccessControl;
using Org.IdentityConnectors.Framework.Spi.Operations;
using Org.IdentityConnectors.Framework.Common.Objects;
using Org.IdentityConnectors.Common;

namespace Org.IdentityConnector.FlatFileConnector
{
    /// <summary>
    /// FlatFileConnector showing implementation of SchemaOp, test, create,
delete, update and search operations.
    /// </summary>

[ConnectorClass("FlatFileConnector_DisplayNameKey",typeof(FlatFileConfiguration))]
    public class FlatFileConnector :
PoolableConnector,CreateOp,SchemaOp,TestOp,DeleteOp,UpdateOp,SearchOp<String>
    {

        /// <summary>
        /// Flat file configuration instance. This instance has the target system
information.
        /// </summary>
        private FlatFileConfiguration config;


        #region Init
        /// <summary>
        /// Create a connection to target and store it for later use. But here we
just set attributes of target file
        /// name to Normal
        /// </summary>
        /// <param name="config">Configuration Object</param>
        public void Init(Configuration config)
        {
            this.config = (FlatFileConfiguration)config;
            File.SetAttributes(this.config.FileName, FileAttributes.Normal);
        }
        #endregion



        #region CreateOp Members
        /// <summary>
        /// This creates a new row in the target file with the data as sent in the 
'attrs'
        /// </summary>
        /// <param name="objClass">The ObjectClass. Here we support only 
Account</param>
        /// <param name="attrs">Attributes of this Account that need to be created 
on target</param>
        /// <param name="options">Will always be empty</param>
        /// <returns>Unique id Uid, representing the Account which was just 
created</returns>
        public Uid Create(ObjectClass objClass, ICollection<ConnectorAttribute> 
attrs, OperationOptions options)
        {
            ConnectorAttribute NameAttribute = 
ConnectorAttributeUtil.Find(Name.NAME, attrs);
            ConnectorAttribute AgeAttribute = ConnectorAttributeUtil.Find("Age", 
attrs);
            ConnectorAttribute QualificationAttribute = 
ConnectorAttributeUtil.Find("Qualification", attrs);
            ConnectorAttribute GenderAttributute = 
ConnectorAttributeUtil.Find("Gender", attrs);
            StreamWriter writer = File.AppendText(this.config.FileName);
            writer.WriteLine("\nName:" + 
ConnectorAttributeUtil.GetAsStringValue(NameAttribute) + this.config.Delimiter + 
"Age:" + ConnectorAttributeUtil.GetAsStringValue(AgeAttribute) + 
this.config.Delimiter + "Qualification:" + 
ConnectorAttributeUtil.GetAsStringValue(QualificationAttribute) + 
this.config.Delimiter + "Gender:" + 
ConnectorAttributeUtil.GetAsStringValue(GenderAttributute));
            writer.Flush();
            writer.Dispose();
            writer.Close();
            return new 
Uid(ConnectorAttributeUtil.GetAsStringValue(NameAttribute));
        }
        #endregion



        #region DeleteOp Members
        /// <summary>
        /// Deletes an entity from target flat file. We support only ACCOUNT 
object class.
        /// If the Uid (user name) is not found then UnknownUidException is thrown
        /// </summary>
        /// <param name="objClass"></param>
        /// <param name="uid"></param>
        /// <param name="options"></param>
        public void Delete(ObjectClass objClass, Uid uid, OperationOptions 
options)
```

```
        {
           String[] allLines = File.ReadAllLines(this.config.FileName);
            String[] newLines = new String[allLines.Length];
            Boolean userExisted = false;
            for (int i = 0; i < allLines.Length; i++)
            {
                char[] separator = new char[] { '$' };
               String[] thisLineSplit = allLines[i].Split(separator);

                String name = "";
                foreach (String str in thisLineSplit)
                {
                    if (str.StartsWith("Name"))
                    {
                        name = str;
                        break;
                    }
                }
                if (!name.Equals("Name" + ":" + uid.GetUidValue()))
                {
                    newLines[i] = allLines[i];
                }
                else
                {
                    userExisted = true;
                }

            }
            if (userExisted)
            {
                File.WriteAllText(this.config.FileName, String.Empty);
                File.WriteAllLines(this.config.FileName, newLines);
            }
            else
            {
                throw new UnknownUidException("Uid "+uid.GetUidValue()+" not
found");
            }
        }
        #endregion

        #region UpdateOp Members
        /// <summary>
        /// Updates information of an existing user on the target flat file
        /// </summary>
        /// <param name="objclass">The ObjectClass. Here we support only
user</param>
        /// <param name="uid">Unique id of the user using which we can find out
the user on target. This is the returned vaue by CreateOp implementation</param>
        /// <param name="replaceAttributes">Updated attributes of user which
should replace all existing user information on target</param>
        /// <param name="options">This will always be empty</param>
        /// <returns>Updated uid. It can be the same value which was provided to
this method.</returns>
        public Uid Update(ObjectClass objclass, Uid uid,
ICollection<ConnectorAttribute> replaceAttributes, OperationOptions options)
        {
            String uidValue = uid.GetUidValue();
            String[] allLines = File.ReadAllLines(this.config.FileName);
            String[] updatedLines = new String[allLines.Length];
            Boolean userExists = false;
            Uid updatedUid = uid;
            for(int i = 0; i < allLines.Length; i++)
            {
                String[] thisLineSplit = allLines[i].Split(new char[] { '$' });
                String name = "";
                foreach (String str in thisLineSplit)
                {
                    if (str.StartsWith("Name"))
                    {
                        name = str;
                        break;
                    }
                }
                String nameToBeUpdated = "Name:" + uidValue;
                if (!name.Equals(nameToBeUpdated))
                {
                    updatedLines[i] = allLines[i];
                }
                else
                {
                    ConnectorAttribute NameAttribute =
ConnectorAttributeUtil.Find(Name.NAME, replaceAttributes);
                    ConnectorAttribute AgeAttribute =
ConnectorAttributeUtil.Find("Age", replaceAttributes);
                    ConnectorAttribute QualificationAttribute =
ConnectorAttributeUtil.Find("Qualification", replaceAttributes);
                    ConnectorAttribute GenderAttribute =
ConnectorAttributeUtil.Find("Gender", replaceAttributes);
                    updatedLines[i] =
"Name:"+NameAttribute.Value.First().ToString()+this.config.Delimiter+

AgeAttribute.Name+":"+AgeAttribute.Value.First().ToString()+this.config.Delimiter+

QualificationAttribute.Name+":"+QualificationAttribute.Value.First().ToString()+th
is.config.Delimiter+

GenderAttribute.Name+":"+GenderAttribute.Value.First().ToString();
                    userExists = true;
                    updatedUid = new Uid(NameAttribute.Value.First().ToString());
                }

            }
            File.WriteAllText(this.config.FileName, String.Empty);
            File.WriteAllLines(this.config.FileName, updatedLines);
            if (!userExists)
            {
                throw new UnknownUidException("User "+uid.GetUidValue()+" not
found");
            }
            return updatedUid;
        }

        #endregion

        #region SearchOp<string> Members

        /// <summary>
        /// Returns a filter translator used by ExecuteQuery. The functionality of
filter translator is to translate any filters provided by calling application
```

```
(OIM/OW/OPAM) to native queries.
        /// </summary>
        /// <param name="oclass">The ObjectClass. We support only ACCOUNT</param>
        /// <param name="options">Options</param>
        /// <returns>FilterTranslator instance</returns>

        public
Org.IdentityConnectors.Framework.Common.Objects.Filters.FilterTranslator<string>
CreateFilterTranslator(ObjectClass oclass, OperationOptions options)
        {
            return new FlatFileFilterTranslator();
        }

        /// <summary>
        /// Performs search on target based on query. Uses the handler instance to
return back the searched result.
        /// </summary>
        /// <param name="oclass">The ObjectClass. This tells if we have to search
for user (ACCOUNT) or group (GROUP). We support only user</param>
        /// <param name="query">Query as returned by FilterTranslator</param>
        /// <param name="handler">handler to return back result to caller</param>
        /// <param name="options">Options containing what attributes of entity to
return back</param>
        public void ExecuteQuery(ObjectClass oclass, string query, ResultsHandler
handler, OperationOptions options)
        {

            String[] results = GetResults(query);
            foreach (String result in results)
            {
                Console.WriteLine("Result = "+result);
                String result1 = result.Trim();
                if (result1.Length > 0)
                {
                    Console.WriteLine("Submitting result = " + result1);
                    SubmitConnectorObject(result1, handler);
                }
            }
        }

        #region SchemaOp Members
        /// <summary>
        /// Defines the schema supported by this connector
        /// </summary>
        /// <returns>Schema</returns>
        public Schema Schema()
        {
            SchemaBuilder schemaBuilder = new
SchemaBuilder(SafeType<Connector>.Get(this));
            ICollection<ConnectorAttributeInfo> connectorAttributeInfos = new
List<ConnectorAttributeInfo>();

connectorAttributeInfos.Add(ConnectorAttributeInfoBuilder.Build("Name"));

connectorAttributeInfos.Add(ConnectorAttributeInfoBuilder.Build("Age"));

connectorAttributeInfos.Add(ConnectorAttributeInfoBuilder.Build("Qualification"));

connectorAttributeInfos.Add(ConnectorAttributeInfoBuilder.Build("Gender"));
            schemaBuilder.DefineObjectClass(ObjectClass.ACCOUNT_NAME,
connectorAttributeInfos);
            return schemaBuilder.Build();
        }

        #endregion

        #region TestOp Members
        /// <summary>
        /// Should ideally test the connecttion with target. But here we just
print something as we have assumed that target file is on same machine
        /// </summary>
        public void Test()
        {
            Console.Write("Tested connection!");
        }

        #endregion


        #region CheckAlive
        /// <summary>
        /// Check connection to target system is alive or not. But here we just
check if target file name
        /// provided in the FlatFileConfiguration is available or not.
        /// </summary>
        public void CheckAlive()
        {
            if (!File.Exists(this.config.FileName))
            {
                throw new ConnectorException("Target file " + this.config.FileName
+ " does not exist");
            }
        }
        #endregion



        #region Dispose
        /// <summary>
        /// Remove connection from target, dispose any of the resources used. But
here we just chill.
        /// </summary>
        public void Dispose()
        {
            //chill :)
        }
        #endregion

        private void SubmitConnectorObject(String result, ResultsHandler handler)
        {
            ConnectorObjectBuilder cob = new ConnectorObjectBuilder();
            String[] resultSplit = result.Split(new char[]{'$'});
            ICollection<ConnectorAttribute> attrs = new
List<ConnectorAttribute>();
            foreach (String str in resultSplit)
            {
                ConnectorAttributeBuilder cab = new ConnectorAttributeBuilder();
                cab.AddValue(str.Split(new char[] { ':' })[1]);
                if (str.StartsWith("Name"))
                {
                    cob.SetName(Name.NAME);
                    cob.SetUid(str.Split(new char[] { ':' })[1]);
                    cab.Name = Name.NAME;
                }
                else
                {
                    cab.Name = str.Split(new char[] { ':' })[0];
                }
                attrs.Add(cab.Build());
            }
            cob.AddAttributes(attrs);
            handler(cob.Build());
        }

        private String[] GetResults(String query)
        {
            String[] allLines = File.ReadAllLines(this.config.FileName);
            String[] results = allLines;
if (query != null)
            {
                for (int i = 0; i < allLines.Length; i++)
                {
                    String[] thisLineSplit = allLines[i].Split(new char[]{'$'});
                    Boolean foundResult = false;
                    foreach (String str in thisLineSplit)
                    {
                        if (str.StartsWith("Name") && str.Equals(query))
                        {
                            foundResult = true;
                            break;
                        }
                    }
                    if (foundResult)
                    {
                        return new String[] {allLines[i]};
                    }
                }
            }

            return results;
        }

        #endregion
    }
}
```

4. This connector supports only the CreateEqualsExpression operation. Implement the CreateEqualsExpression. Example 6–3 illustrates the sample implementation of Org.IdentityConnectors.Framework.Common.Objects.Filters.AbstractFilterTranslator<T> that defines the filter operation.

Example 6–3 Implementation of AbstractFilterTranslator<T>

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Org.IdentityConnectors.Framework.Common.Objects.Filters;
using Org.IdentityConnectors.Framework.Common.Objects;
```

```
namespace Org.IdentityConnector.FlatFileConnector
{
    /// <summary>
    /// FlatFileFilterTranslator. This translator converts the equalsFilter
provided by the calling application to native query which can be used by the
connector while searching.
    /// The implementation shown supports only equals filter. i.e it has provided
implementation for only CreateEqualsExpression, this means that if any other
filter is provided
    /// by the calling application, it would not be translated as a native query
and search implementation gets all users and filtering will be done by ICF with
all results.
    ///
    /// </summary>
    public class FlatFileFilterTranslator : AbstractFilterTranslator<String>
    {
        /// <summary>
        /// Creates a native query for equals filter and returns it only if equals
filter is constructed for Name attribute and not for any other attributes.
        /// </summary>
        /// <param name="filter">Filter provided by calling application</param>
        /// <param name="not"></param>
        /// <returns></returns>
        protected override string CreateEqualsExpression(EqualsFilter filter, bool
not)
        {
            ConnectorAttribute attr = filter.GetAttribute();
            if (attr.Name.Equals(Name.NAME))
            {
                return "Name:" + attr.Value.First().ToString();
            }
            return null;
        }
    }
}
```

5. Implement the different classes (as mentioned in steps 2, 3, and 4).
6. Make a note of AssemblyVersion present in the AssemblyInfo.cs of the project.

   Sample AssemblyInfo.cs file:

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("FlatFileConnector")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("Oracle Corporation")]
[assembly: AssemblyProduct("FlatFileConnector")]
[assembly: AssemblyCopyright("Copyright © Oracle Corporation 2012")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
```

```
[assembly: ComVisible(true)]

// The following GUID is for the ID of the typelib if this project is exposed
to COM
[assembly: Guid("79eec317-62bd-49a5-9512-88d61135684c")]

// Version information for an assembly consists of the following four values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Build and Revision
Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

7. Build the project. The project must create the connector DLL.

6.2 Deploying the Identity Connector Bundle on .NET Connector Server

For all the connectors that are implemented in .NET, you need to have .NET Connector Server for the execution of the connector. The connector bundle cannot be deployed within Oracle Identity Manager. Therefore, you must perform the following procedures in order to integrate the ICF .NET Identity Connector with Oracle Identity Manager:

- Section 6.2.1, "Registering the Connector Bundle with .NET Connector Server"
- Section 6.2.2, "Creating Basic Identity Connector Metadata"
- Section 6.2.3, "Creating Provisioning Metadata"
- Section 6.2.4, "Creating Reconciliation Metadata"

6.2.1 Registering the Connector Bundle with .NET Connector Server

For registering or deploying the connector bundle on .NET Connector Server, perform the following steps:

1. Install the .NET Connector Server. See Section 4.6.2.1, "Installing the .NET Connector Server" for more information about installing the .NET Connector Server.
2. Stop the Connector Server. Make sure that Connector Server Service is not running.
3. Copy the connector DLL in the CONNECTOR_SERVER_HOME location. CONNECTOR_SERVER_HOME is the location where ConnectorServer.exe and other connector server related files are present after .NET Connector Server installation.
4. Start the .NET Connector Server.

6.2.2 Creating Basic Identity Connector Metadata

This metadata configuration is needed for both provisioning and reconciliation. Perform the following procedures by using the Oracle Identity Manager Design Console.

- Section 6.2.2.1, "Creating the IT Resource Type Definition"
- Section 6.2.2.2, "Creating the Resource Object"

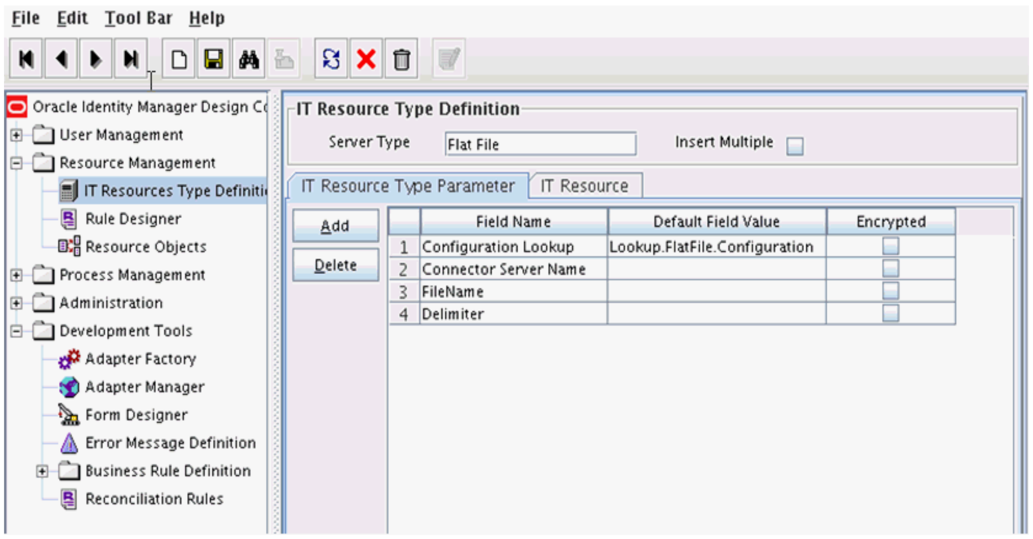
6.2.2.1 Creating the IT Resource Type Definition

An IT resource type definition is the representation of a resource's connection information. The configuration parameters in the IT resource type definition should be matched with the configuration parameters of the connector bundle. The values of the parameters in the IT resource will be set in the bundle configuration.

Note: You may include parameters the bundle configuration is not using. They produce no negative effects on the bundle operations.

1. Log in to the Oracle Identity Manager Design Console.
2. Click **IT Resource Type Definition** under Resource Management.
3. Create a new IT Resource Type Definition with the Server Type defined as Flat File.
4. Add the following parameters as illustrated in Figure 6–1.
   - Configuration Lookup is the marker of the main configuration lookup for the resource. The name of the parameter must be Configuration Lookup. It is a good practice to add a value to Default Field Value.
   - Delimiter maps to the Delimiter parameter in the bundle configuration. The value of this parameter will be passed.
   - FileName maps to the FileName parameter in the bundle configuration. The value of this parameter will be passed.
   - Connector Server Name, provide the connector server IT Resource name where .NET Connector Server is running.

Figure 6–1 IT Resource Type Definition in Design Console



Server Type: Flat File
Insert Multiple: ☐

	Field Name	Default Field Value	Encrypted
1	Configuration Lookup	Lookup.FlatFile.Configuration	☐
2	Connector Server Name		☐
3	FileName		☐
4	Delimiter		☐

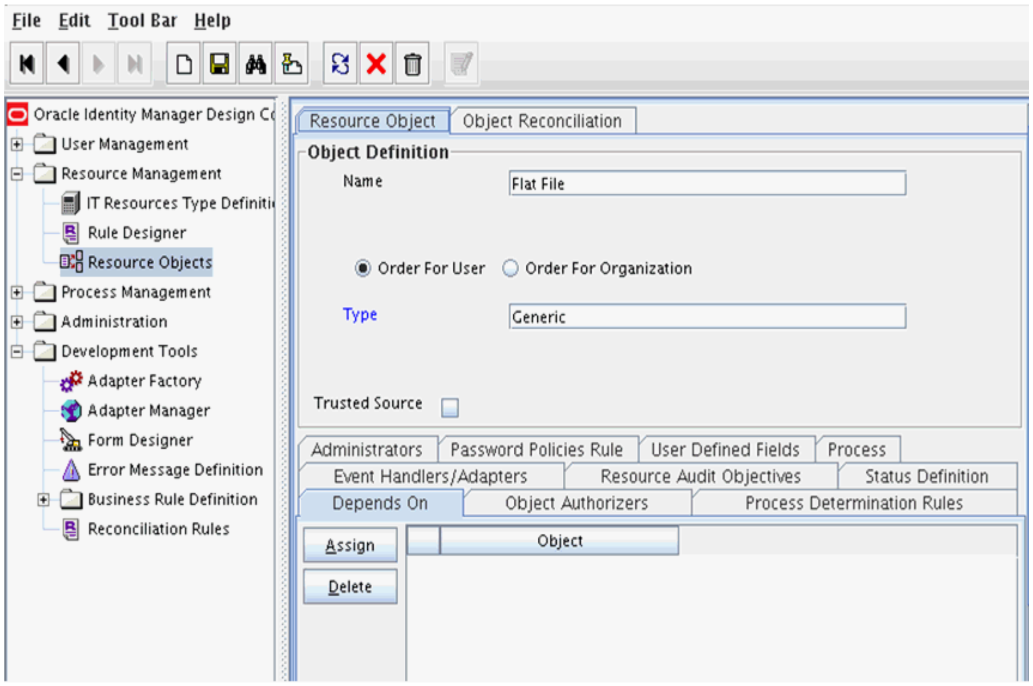
6.2.2.2 Creating the Resource Object

The resource object is the Oracle Identity Manager representation of a resource. The connector bundle is tied to the resource object.

1. Log in to the Oracle Identity Manager Design Console.
2. Click **Resource Objects** under Resource Management.
3. Create a new resource object with the name **Flat File**.

   As the resource object is a target resource, do not check the Trusted Source box as illustrated in Figure 6–2.

Figure 6–2 Resource Objects in Design Console



Name: Flat File
[✓ Order For User] Order For Organization
Type: Generic
Trusted Source: ☐

6.2.2.3 Creating Lookups

Separate lookups have to be defined for different objects supported by the connector bundle. This lookup can contain provisioning and reconciliation related information for those objects. The Main Configuration Lookup is the root for object specific lookups as it contains the pointers to those lookups. The following sections contain information on how to create lookups.

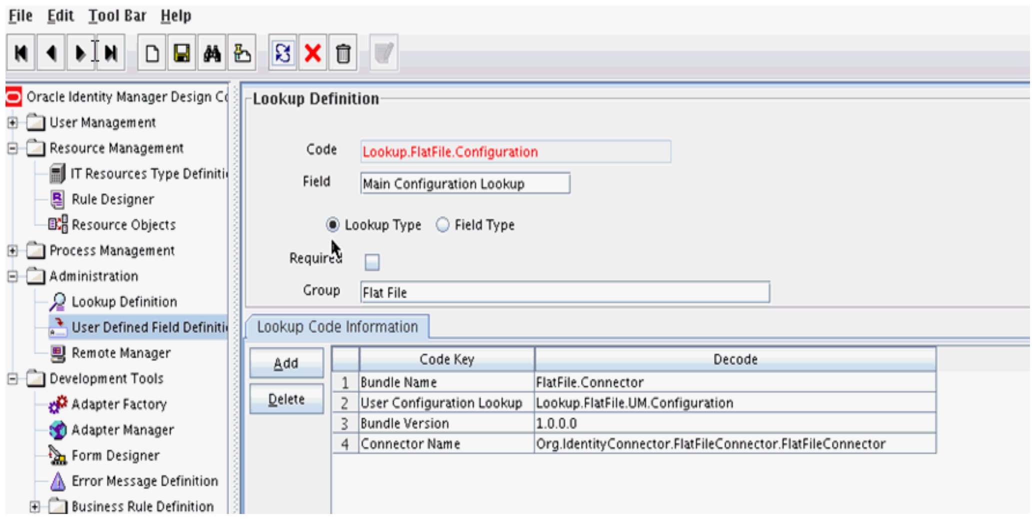
- Creating the Main Configuration Lookup
- Creating Object Type Configuration Lookup

6.2.2.3.1 Creating the Main Configuration Lookup The Configuration Lookup (as defined in Section 6.2.2.1, "Creating the IT Resource Type Definition") holds connector bundle configurations that are not counted as connection information. If a configuration parameter is not found in the IT Resource Type Definition, Oracle Identity Manager will look in the Configuration Lookup. The main Configuration Lookup contains bundle properties and bundle configurations. Bundle Property parameters are mandatory as they are needed for identifying the correct bundle. Bundle configurations that are not defined as part of the IT resource type definition (discussed in Section 6.2.2.1, "Creating the IT Resource Type Definition") can be declared here.

> **Note:** The values for Code Key should match exactly as illustrated. The values for Decode are specific to the connector bundle.

1. Log in to the Oracle Identity Manager Design Console.
2. Click Lookup Definition under Administration.
3. Create a new lookup and add Lookup.FlatFile.Configuration as the value for Code.
4. Add the following Lookup Code Information as illustrated in Figure 6–3.
    - Add *AssemblyVersion* as the required Bundle Version.
    - Add **FlatFile.Connector** as the required Bundle Name. The bundle name can be identified from the connector dll name. Connector DLL is in BUNDLE_NAME.dll format.
    - Add **Org.IdentityConnector.FlatFileConnector.FlatFileConnector** as the required Connector Name.
    - OBJECT_TYPE_NAME Configuration Lookup is the configuration lookup for the particular object type. In this example, the object type is User as User Configuration Lookup is defined.

Figure 6–3 Lookup Definition in Design Console



Code	Lookup.FlatFile.Configuration
Field	Main Configuration Lookup
	[✓ Lookup Type] Field Type
Required	☐
Group	Flat File

	Code Key	Decode
1	Bundle Name	FlatFile.Connector
2	User Configuration Lookup	Lookup.FlatFile.UM.Configuration
3	Bundle Version	1.0.0.0
4	Connector Name	Org.IdentityConnector.FlatFileConnector.FlatFileConnector

6.2.2.3.2 Creating Object Type Configuration Lookup Object type configuration lookup contains the parameters specific to the particular object type. Object type is an entity over which an identity connector operates. It is mapped to ICF ObjectClass. In Section 6.2.2.3.1, "Creating the Main Configuration Lookup," User Configuration Lookup has been referenced so that User is the object type, in this case mapped to ObjectClass.ACCOUNT. (Roles and UserJobData are two other object types.) The object type name has to match with ObjectClass name supported by the identity connector bundle. The User object type is mapped to predefined ObjectClass.ACCOUNT, the Group object type is mapped to predefined ObjectClass.GROUP. If the identity connector supports multiple objects, then this step must be repeated for each.

Note: Because these use cases cover only the basic functionality, the configuration is kept to the mandatory attribute.

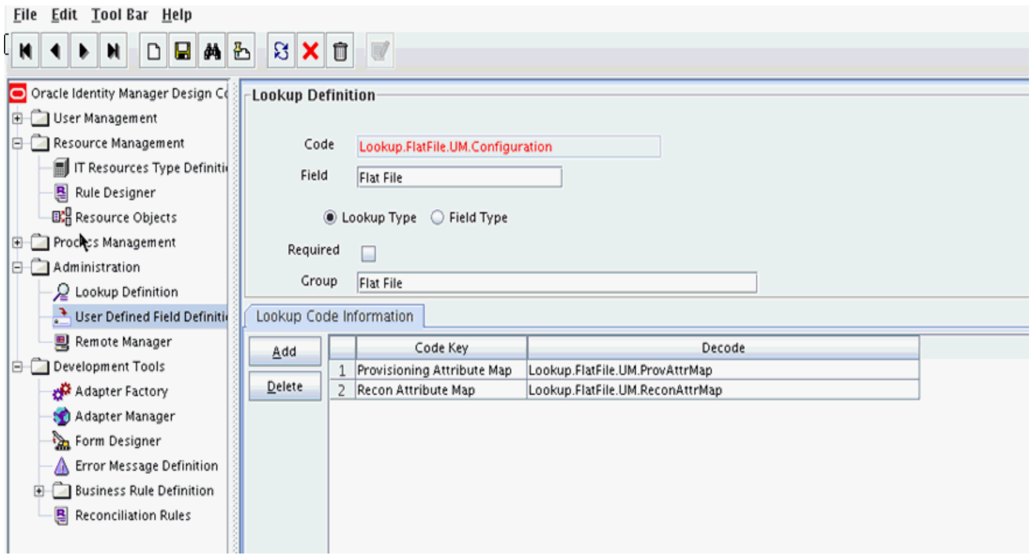
---

1. Log in to the Oracle Identity Manager Design Console.
2. Click Lookup Definition under Administration.
3. Create a new Lookup and add Lookup.FlatFile.UM.Configuration as the Code.
4. Set the following attributes as illustrated in Figure 6–4.

Note: This tutorial focuses on the minimum configurations needed to run an identity connector.

- **Provisioning Attribute Map** takes a value of Lookup.FlatFile.UM.ProvAttrMap. This lookup contains the mapping between Oracle Identity Manager fields and identity connector attributes. The mapping is used during provisioning.
- **Reconciliation Attribute Map** takes a value of Lookup.FlatFile.UM.ReconAttributeMap. This lookup contains the mapping between Oracle Identity Manager reconciliation fields and identity connector attributes. The mapping is used during reconciliation.

Figure 6–4 Second Lookup Definition in Design Console



Lookup Definition

Code: Lookup.FlatFile.UM.Configuration
Field: Flat File
(● Lookup Type) ○ Field Type
Required: ☐
Group: Flat File

	Code Key	Decode
1	Provisioning Attribute Map	Lookup.FlatFile.UM.ProvAttrMap
2	Recon Attribute Map	Lookup.FlatFile.UM.ReconAttrMap

6.2.3 Creating Provisioning Metadata

The following sections should be followed in order to configure Oracle Identity Manager for flat file provisioning.

- Section 6.2.3.1, "Creating a Process Form"
- Section 6.2.3.2, "Creating Adapters"
- Section 6.2.3.3, "Creating A Process Definition"
- Section 6.2.3.4, "Creating a Provisioning Attribute Mapping Lookup"

6.2.3.1 Creating a Process Form

A process form is used as the representation of object attributes on Oracle Identity Manager. This facilitates user input to set object attributes before passed to the connector bundle for an operation.

Attributes defined in the process form are not conventions. The form is a way to challenge the attributes that need to be passed to the identity connector. In general, define an attribute for each supported attribute in the identity connector.

Note: It is good practice to have a one to one mapping on the identity connector attributes.

There should be a field for querying the IT resource that should be associated with the respective IT Resource Type Definition. Variable type of each field should map to the type of the object attribute.

1. Log in to the Oracle Identity Manager Design Console.
2. Click **Form Designer** under Development Tools.
3. Create a new form with the Table Name UD_FLATFILE as illustrated in Figure 6–5.

Figure 6–5 Form Designer in Design Console

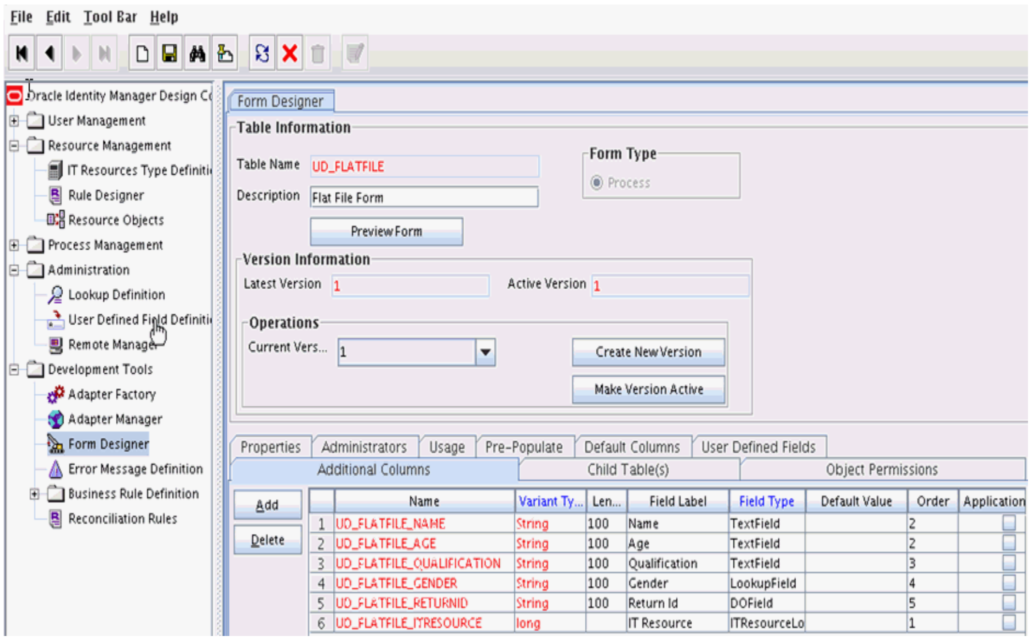


Table Information

Table Name: UD_FLATFILE

Description: Flat File Form

Form Type: Process

Version Information

Latest Version: 1

Active Version: 1

Operations

Current Vers...: 1

	Name	Variant Ty...	Len...	Field Label	Field Type	Default Value	Order	Application
1	UD_FLATFILE_NAME	String	100	Name	TextField		2	☐
2	UD_FLATFILE_AGE	String	100	Age	TextField		2	☐
3	UD_FLATFILE_QUALIFICATION	String	100	Qualification	TextField		3	☐
4	UD_FLATFILE_GENDER	String	100	Gender	LookupField		4	☐
5	UD_FLATFILE_RETURNID	String	100	Return Id	DOField		5	☐
6	UD_FLATFILE_ITRESOURCE	long		IT Resource	ITResourceLo		1	☐

4. Add the attributes defined in the connector schema, as listed in Table 6–1.

Table 6–1 Form Designer Fields

Name	Variant	Field Label	Field Type
UD_FLATFILE_NAME	String	Name	TextField
UD_FLATFILE_AGE	String	Age	TextField
UD_FLATFILE_QUALIFICATION	String	Qualification	TextField
UD_FLATFILE_GENDER	String	Gender	LookupField
UD_FLATFILE_RETURNIDQ	String	Return Id	DOField
UD_FLATFILE_ITRESOURCE	Long	IT Resource	ITResourceLookup

Note: The flat file column names are FirstName, ChangeNo, EmailID, Server, LastName, and AccountID.

---

5. Click the Properties tab.
6. Add the following properties to Server(ITResourceLookupField) as illustrated in Figure 6–6.
   - Required = true
   - Type = Flat File

Figure 6–6 Properties of Form Designer in Design Console

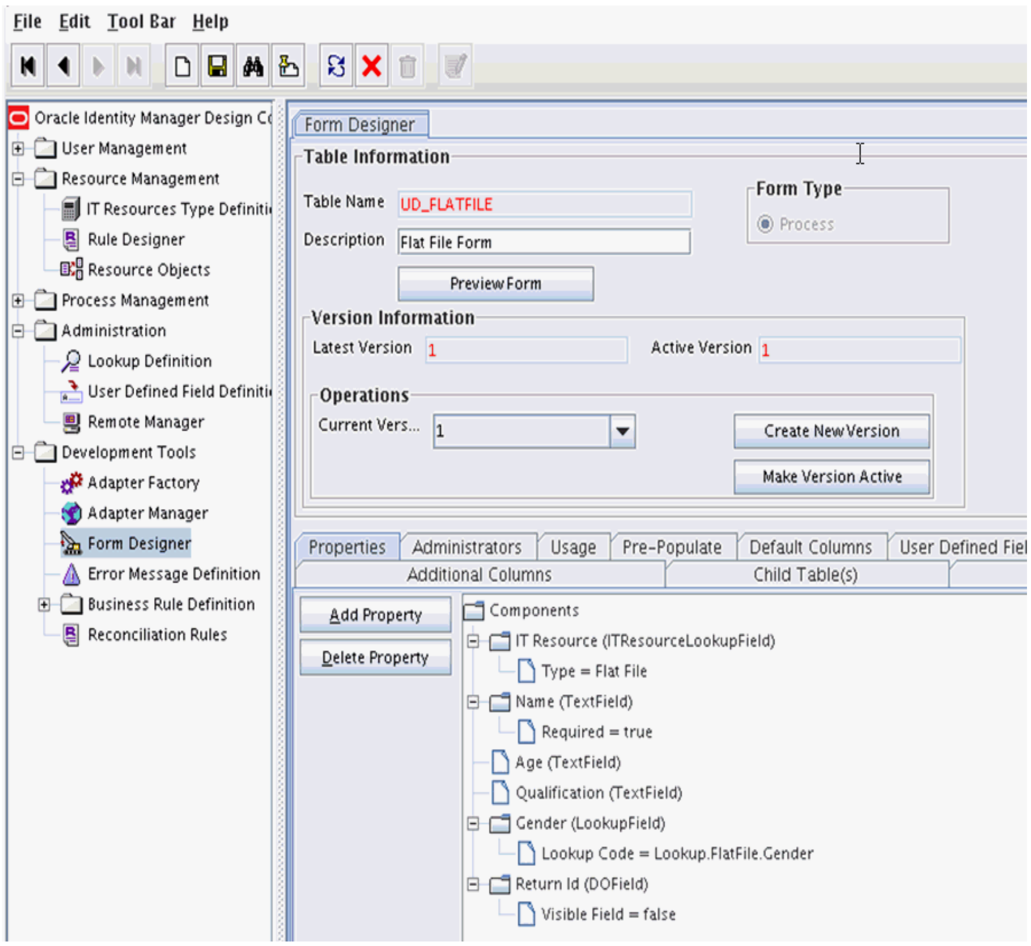


Table Name: UD_FLATFILE
Description: Flat File Form
Latest Version: 1
Active Version: 1
Current Vers...: 1

```
Components
  IT Resource (ITResourceLookupField)
    Type = Flat File
  Name (TextField)
    Required = true
  Age (TextField)
  Qualification (TextField)
  Gender (LookupField)
    Lookup Code = Lookup.FlatFile.Gender
  Return Id (DOField)
    Visible Field = false
```

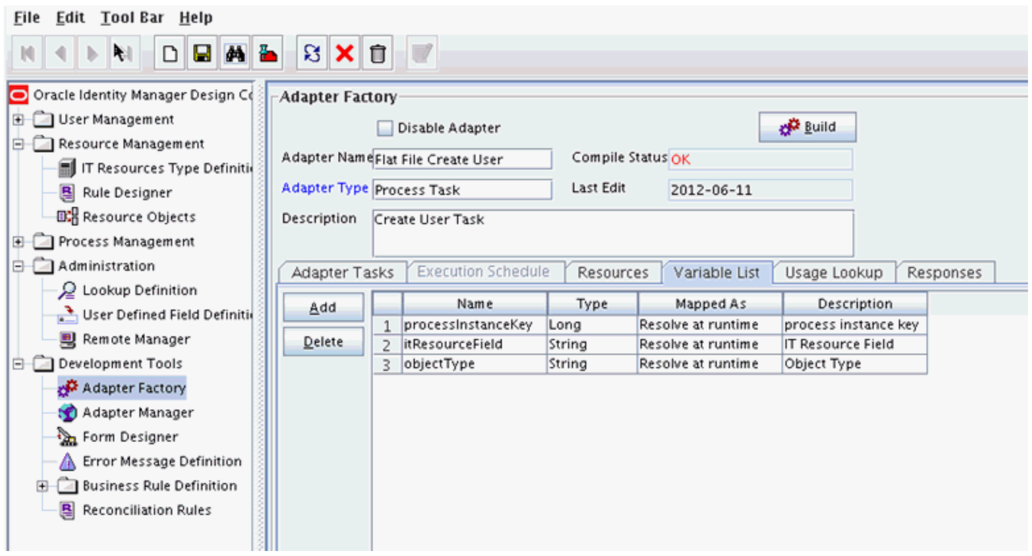
7. Save the form.
8. Click Make Version Active.

6.2.3.2 Creating Adapters

An adapter has to be created for all operations supported by the connector bundle, including Create, Update, and Delete.

1. Log in to the Oracle Identity Manager Design Console.
2. Click **Adapter Factory** under Development Tools.
3. Create a new adapter and add Flat File Create User as the Adapter Name.
4. Add Process Task as the Adapter Type.
5. Save the adapter.
6. Click the **Variable List** tab and add the following variables, as shown in Figure 6–7.
   - objectType with Type String and Mapped as Resolve at runtime.
   - processInstanceKey with Type long and Mapped as Resolve at runtime.
   - itResourceFieldName with Type String and Mapped as Resolve at runtime.

Figure 6–7 Adapter Factory Variable List in Design Console



Disable Adapter

Adapter Name: Flat File Create User

Compile Status: OK

Adapter Type: Process Task

Last Edit: 2012-06-11

Description: Create User Task

	Name	Type	Mapped As	Description
1	processInstanceKey	Long	Resolve at runtime	process instance key
2	itResourceField	String	Resolve at runtime	IT Resource Field
3	objectType	String	Resolve at runtime	Object Type

7. Add a Java functional task to the adapter by following this sub procedure, as shown in Figure 6–8.

   a. Click the **Adapter Tasks** tab.

   b. Select the adapter and click Add.

   c. Select Java from the task options.

   d. Select **icf-oim-intg.jar** from the API source.

   e. Select **oracle.iam.connetors.icfcommon.prov.ICProvisioninManager** as the API Source.

   f. Select **createObject** as the method for the task.

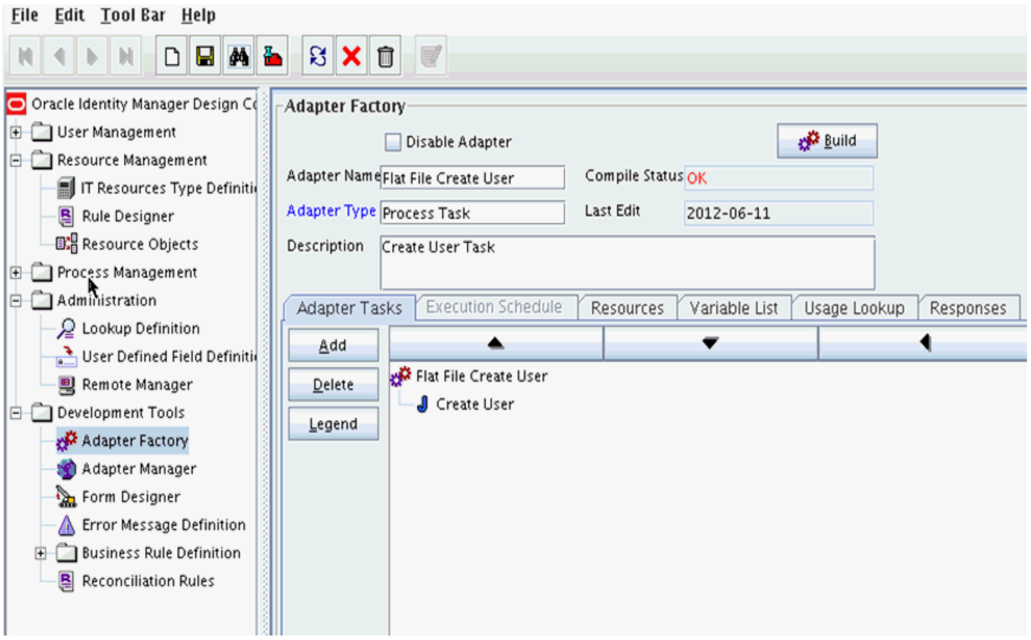
   g. Save the configurations.

   h. Map the variables (previously added to the Variables List) against the appropriate method inputs and outputs.

   i. Map the configuration parameters against the appropriate method inputs and outputs.

      Database Reference maps to Database Reference (Adapter References) and Return Variable maps to Return Variable (Adapter Variables).

Figure 6–8 Adapter Factory in Design Console



Disable Adapter

Adapter Name: Flat File Create User
Compile Status: OK
Adapter Type: Process Task
Last Edit: 2012-06-11
Description: Create User Task

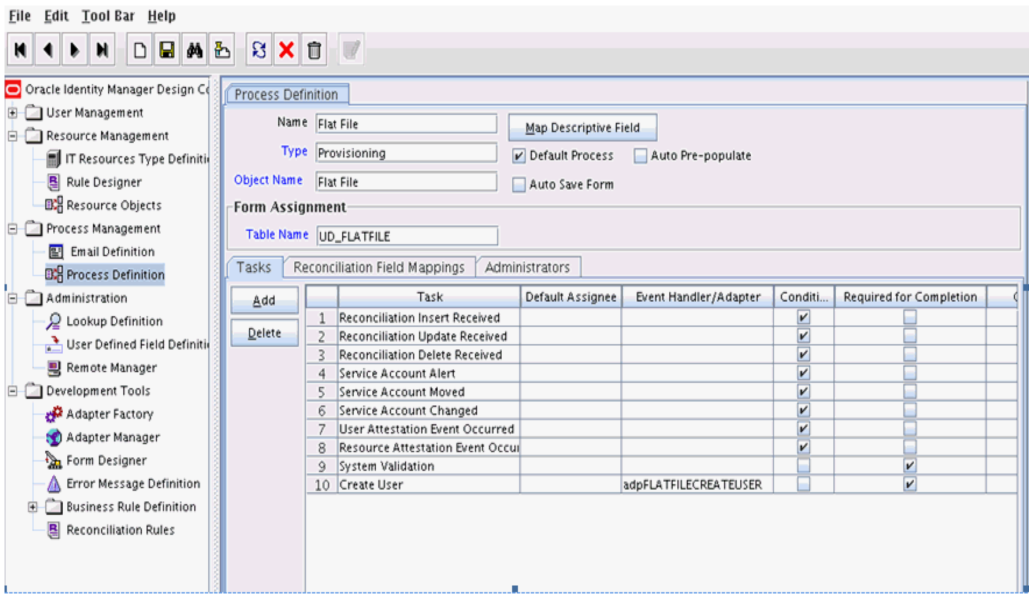
8. Save and build the adapter.

6.2.3.3 Creating A Process Definition

Process Definition defines the behavior of the connector bundle for a particular operation. Every operation has a corresponding task associated with it. This procedure will configure the process definition and integration of the process task for the Create operation.

1. Log in to the Oracle Identity Manager Design Console.
2. Click Process Definition under the Process Management tab.
3. Create a new process definition and name it Flat File as illustrated in Figure 6–9.

Figure 6–9 Process Definition in Design Console



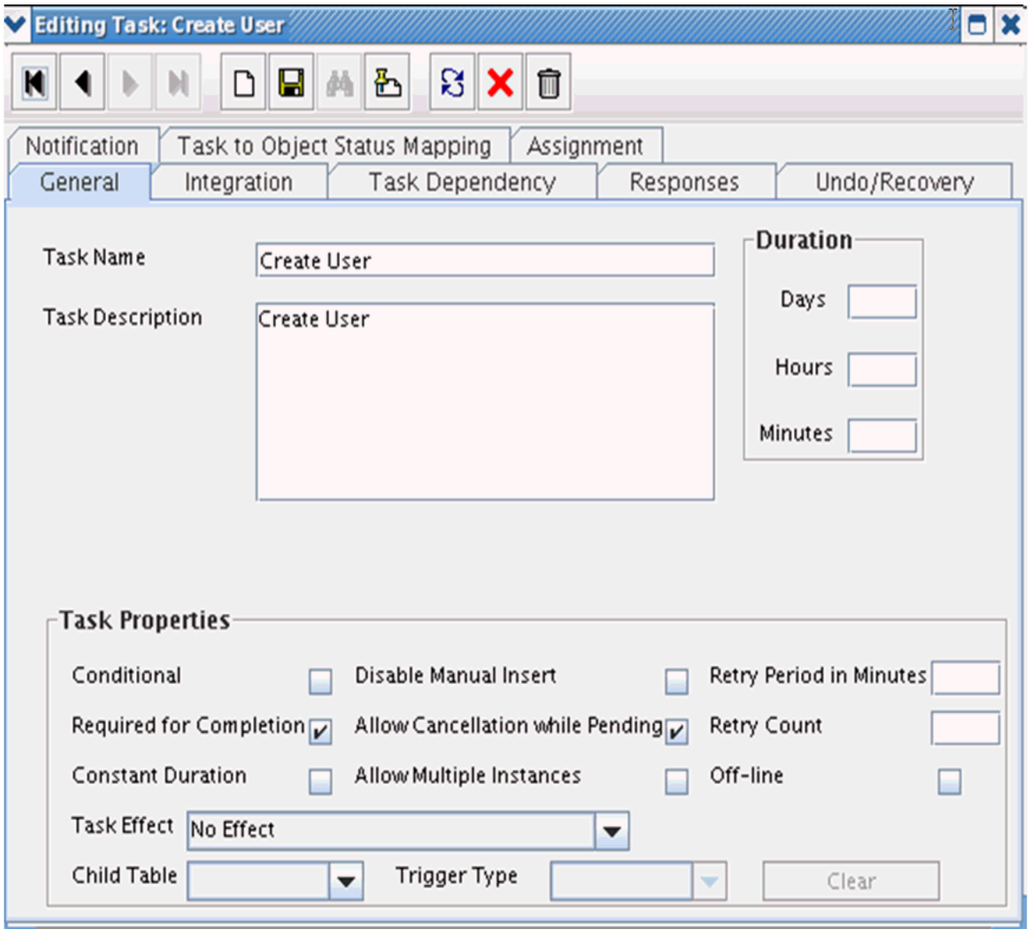
Name: Flat File
Type: Provisioning
Object Name: Flat File
[✓] Default Process
[ ] Auto Pre-populate
[ ] Auto Save Form

Form Assignment
Table Name: UD_FLATFILE

	Task	Default Assignee	Event Handler/Adapter	Conditi...	Required for Completion
1	Reconciliation Insert Received			✓	
2	Reconciliation Update Received			✓	
3	Reconciliation Delete Received			✓	
4	Service Account Alert			✓	
5	Service Account Moved			✓	
6	Service Account Changed			✓	
7	User Attestation Event Occurred			✓	
8	Resource Attestation Event Occu			✓	
9	System Validation				✓
10	Create User		adpFLATFILECREATEUSER		✓

4. Select Provisioning as the Type of process.
5. Provide the resource Object Name for the identity connector; in this example, Flat File.
6. Provide the process form Table Name; in this example, UD_FLATFILE.
7. Add a process task and name it Create User.
8. Double click **Create User** to edit as illustrated in Figure 6–10.

Figure 6–10 Editing Task Screen in Design Console



Task Name: Create User

Task Description: Create User

Conditional	☐	Disable Manual Insert	☐	Retry Period in Minutes	
Required for Completion	✓	Allow Cancellation while Pending	✓	Retry Count	
Constant Duration	☐	Allow Multiple Instances	☐	Off-line	☐

Task Effect: No Effect

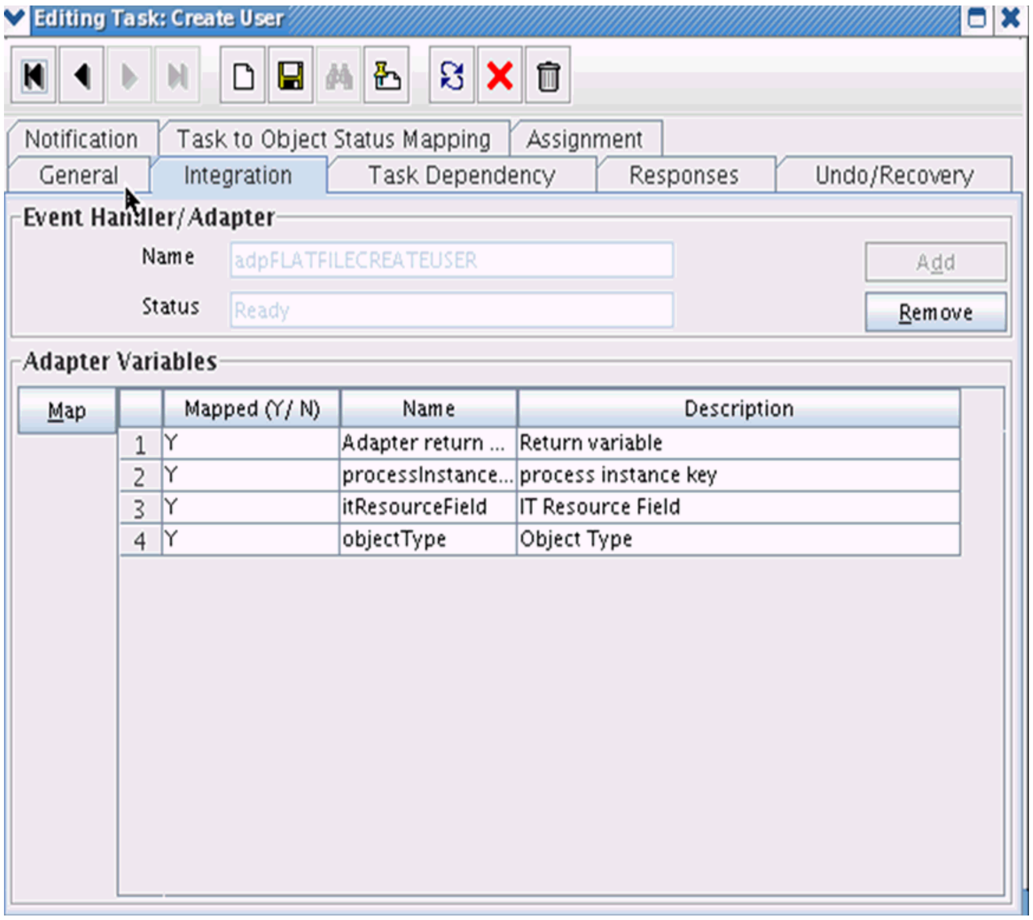
Child Table:

Trigger Type:

9. Click the **Integration** tab.
10. Click Add and select the adpFLATFILECREATEUSER from the list as illustrated in Figure 6–11.

    The adapter will be available only after it is compiled.

Figure 6–11 Integration Tab in Design Console



Name: adpFLATFILECREATEUSER

Status: Ready

	Mapped (Y/ N)	Name	Description
1	Y	Adapter return ...	Return variable
2	Y	processInstance...	process instance key
3	Y	itResourceField	IT Resource Field
4	Y	objectType	Object Type

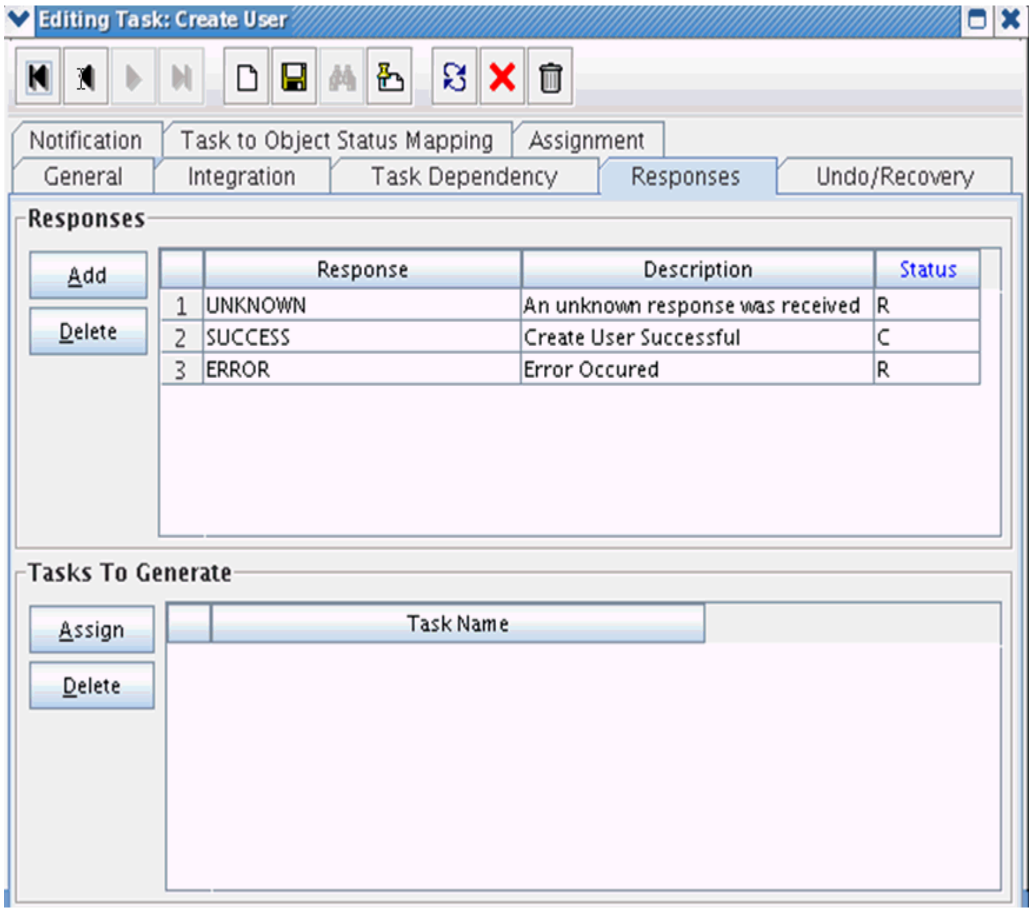
11. Map the variables as follows to set the response code returned by the identity connector.

- Adapter Return Variable – Response Code
- Object Type – [Literal:String] User (Name of the object type)
- Process Instance Key – [Process Data] Process Instance
- IT Resource Field Name – [Literal:String] UD_FLATFILE_ITRESOURCE (Form field name that contains the IT resource information)

12. Click the Responses tab and configure the responses as illustrated in Figure 6–12.

- UNKNOWN can be described as *Unknown response received* with a status of R (Rejected).
- SUCCESS can be described as *Operation completed* with a status of C (Completed).
- ERROR can be described as *Error occurred* with a status of R.

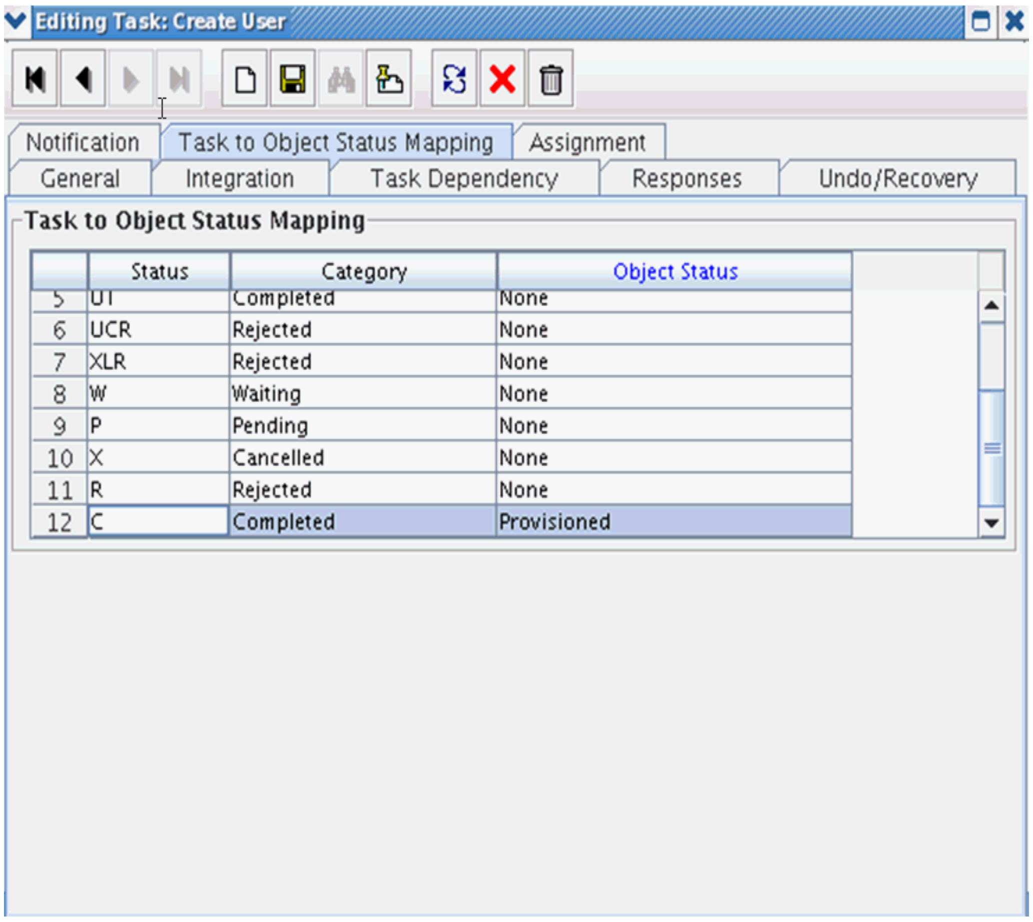
Figure 6–12 Configure Responses in Design Console



	Response	Description	Status
1	UNKNOWN	An unknown response was received	R
2	SUCCESS	Create User Successful	C
3	ERROR	Error Occured	R

13. Click the **Task to Object Status Mapping** tab.
14. Update the Object Status to **Provisioned** for Status C, as shown in Figure 6–13:

Figure 6–13 Task to Object Status Mapping


Editing Task: Create User
Notification
Task to Object Status Mapping
Assignment
General
Integration
Task Dependency
Responses
Undo/Recovery
Task to Object Status Mapping

	Status	Category	Object Status
5	UT	Completed	None
6	UCR	Rejected	None
7	XLR	Rejected	None
8	W	Waiting	None
9	P	Pending	None
10	X	Cancelled	None
11	R	Rejected	None
12	C	Completed	Provisioned

15. Save the process task.

6.2.3.4 Creating a Provisioning Attribute Mapping Lookup

Provisioning Attribute Mapping Lookup contains mappings of Oracle Identity Manager fields to identity connector bundle attributes. In the Provisioning Attribute Mapping Lookup:

- Code keys are Field Labels of the process form.
- Decodes are identity connector bundle attributes.
- Child form attributes can be configured as embedded objects in inputs.
- The identity connector's provisioning operation returns the UID in response. This can be set in a form field by coding it against the identity connector bundle attribute.

Following is the procedure to create a Provisioning Attribute Mapping Lookup.

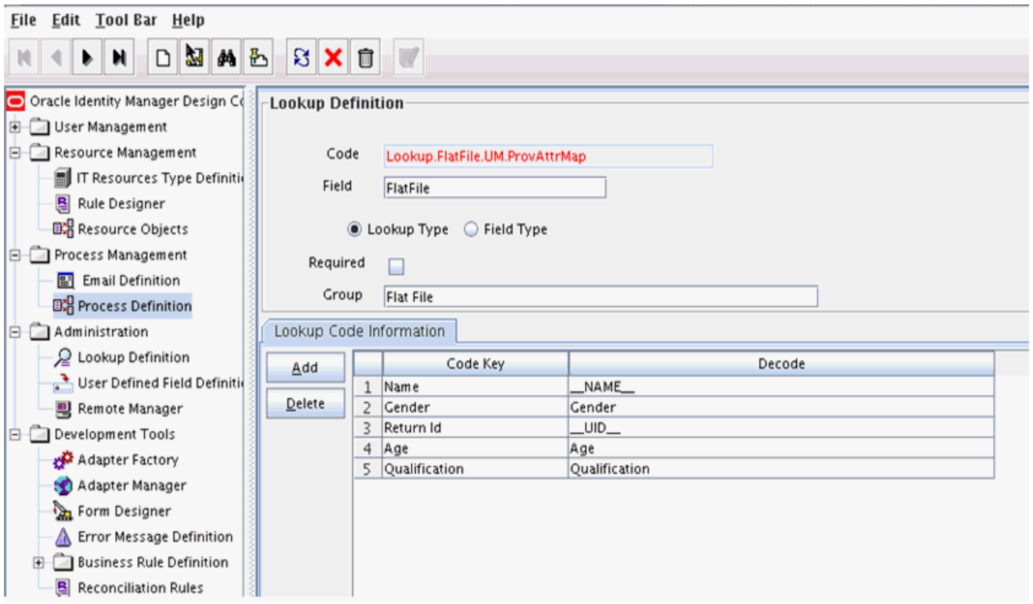
1. Log in to the Oracle Identity Manager Design Console.
2. Click **Lookup Definition** under the Administration tab.
3. Create a new lookup and name it Lookup.FlatFile.UM.ProvAttrMap.

   The name of this lookup is referred from the object type configuration lookup. See Section 6.2.2.3.2, "Creating Object Type Configuration Lookup."

4. Add the form Field Labels as the code keys and identity connector bundle attributes as the decode as shown in Figure 6–14.

   - Name : __NAME__
   - Gender: Gender
   - Return Id: __UID__
   - Age: Age
   - Qualification: Qualification

Figure 6–14 Lookup Code Mapping



Field	Value
Code	Lookup.FlatFile.UM.ProvAttrMap
Field	FlatFile
Type	(● Lookup Type) ○ Field Type
Required	☐
Group	Flat File

	Code Key	Decode
1	Name	__NAME__
2	Gender	Gender
3	Return Id	__UID__
4	Age	Age
5	Qualification	Qualification

6.2.3.4.1 Field Flags Used in the Provisioning Attributes Map

Note: These properties are advanced options and can be skipped for the current implementation of the connector.

For provisioning attributes mapping, the following field flags can be appended to the code key:

- **LOOKUP:** This must be specified for all fields whose values are obtained by running a lookup reconciliation job. The values obtained from lookup reconciliation job have IT Resource Name/Key appended to it. Specifying this flag helps ICF integration to remove the appended value just before passing them onto the bundle. For example, the code key for a field with label Database whose value is obtained by running a lookup reconciliation job looks similar to Database[LOOKUP].

Note: The LOOKUP flag can be specified for both Provisioning and Reconciliation Attribute Map. For provisioning, IT Resource Name/IT Resource Key prefix must be removed. For reconciliation, IT Resource Name/IT Resource Key prefix must be added.

- **IGNORE:** This must be specified for all fields whose values are to be ignored and not sent to bundle. For example, the code key for a field with label Database whose value need not be sent to bundle looks similar to Database[IGNORE].
- **WRITEBACK:** This must be specified for all fields whose values need to be written back into the process form right after the create or update operation. Adding this flag makes the ICF integration layer call ICF Get API to get values of attributes marked with the WRITEBACK flag. For example, the code key for a field with label Database whose value needs to be written back to the process form right after create/update looks similar to Database[WRITEBACK]. For this to work, the connector must implement the GetApiOp interface and provide an implementation for the ConnectorObject getObject(ObjectClass objClass,Uid uid,OperationOptions options) API. This API searches the target for the account whose Uid is equal to the passed in Uid, and builds a connector object containing all the attributes (and their values) that are to be written back to process form.

  Note: If the connector does not implement the GetApiOp interface, then the WRITEBACK flag does not work and an error is generated.

- **DATE:** This must be specified for fields whose type need to be considered as Date, without which the values are considered as normal strings. For example, the code key for a field with label Today whose value needs to be displayed in the date format looks similar to Today[DATE].
- **PROVIDEONPSWDCHANGE:** This must be specified for all fields that need to be provided to the bundle(target) when a password update happens. Some targets expect additional attributes to be specified on every password change. Specifying the PROVIDEONPSWDCHANGE flag, tells ICF integration to send all the extra fields or attributes whenever a password change is requested. For example, the code key for a field with label Extra Attribute Needed for Password Change whose value needs to be provided to bundle(target) while password update looks similar to Extra Attribute Needed for Password Change[PROVIDEONPSWDCHANGE].

6.2.4 Creating Reconciliation Metadata

This section contains the procedures to configure the reconciliation of records from the flat file. You use the target reconciliation as an example; trusted reconciliation can also be configured in a similar fashion. Do the procedures in the listed order.

- Section 6.2.4.1, "Creating a Reconciliation Schedule Task"
- Section 6.2.4.2, "Creating a Reconciliation Profile"
- Section 6.2.4.3, "Setting a Reconciliation Action Rule"
- Section 6.2.4.4, "Creating Reconciliation Mapping"
- Section 6.2.4.5, "Defining a Reconciliation Matching Rule"

6.2.4.1 Creating a Reconciliation Schedule Task

By default, reconciliation uses a Search operation on the connector bundle. This operation is invoked with a schedule task configured using Oracle Identity Manager. This procedure is comprised of the following sub procedures.

1. Section 6.2.4.1.1, "Defining the Schedule Task"
2. Section 6.2.4.1.2, "Creating a Scheduled Job"

6.2.4.1.1 Defining the Schedule Task To define the scheduled task:

1. Create a Deployment Manager XML file containing the scheduled task details as shown in Example 6–4. Make sure to update database value to your database.

Example 6–4 Deployment Manager XML with Scheduled Task Details

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<xl-ddm-data version="2.0.1.0" user="XELSYSADM"
database="jdbc:oracle:thin:@localhost:5524/estView.regress.rdbms.dev.mycompany.com
" exported-date="1307546406635" description="FF">
<scheduledTask repo-type="MDS" name="Flat File Connector User Reconciliation"
mds-path="/db" mds-file="Flat File Connector User Reconciliation.xml">
    <completeXml>
        <scheduledTasks xmlns="http://xmlns.oracle.com/oim/scheduler">
            <task>
            <name>Flat File Connector User Reconciliation</name>
            <class>oracle.iam.connectors.icfcommon.recon.SearchReconTask</class>
            <description>Flat File Connector User Reconciliation</description>
            <retry>0</retry>
            <parameters>
              <string-param required="false" encrypted="false"
helpText="Filter">Filter</string-param>
              <string-param required="false" encrypted="false"
helpText="Incremental Recon Date Attribute">Incremental Recon Date
Attribute</string-param>
              <string-param required="false" encrypted="false" helpText="IT
Resource Name">IT Resource Name</string-param>
              <string-param required="false" encrypted="false" helpText="Object
Type">Object Type</string-param>
              <string-param required="false" encrypted="false" helpText="Latest
Token">Latest Token</string-param>
              <string-param required="false" encrypted="false" helpText="Resource
Object Name">Resource Object Name</string-param>
            </parameters>
          </task>
        </scheduledTasks>
    </completeXml>
</scheduledTask>
</xl-ddm-data>
```

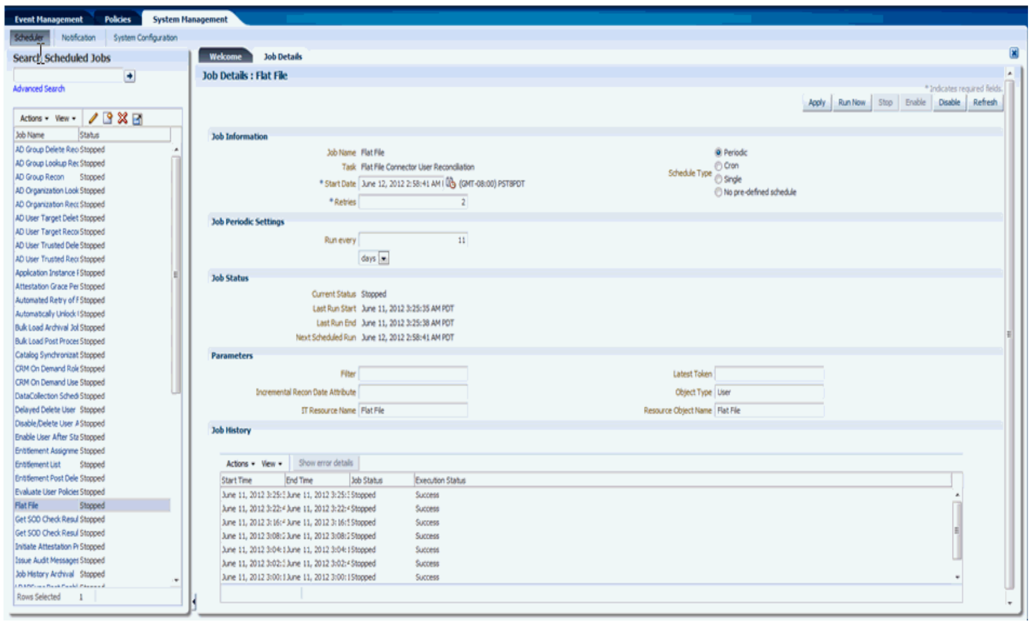
2. Save the file as Flat File Connector User Reconciliation.xml.
3. Login into the Identity System Administration. Under System Management, click **Import**.
4. Select the Flat File Connector User Reconciliation.xml file, and click **Import**.
5. Complete the steps in the wizard.

6.2.4.1.2 Creating a Scheduled Job This procedure explains how to create a scheduled task.

1. Log in to the Oracle Identity Manager Advanced Administration.
2. Click **Scheduler** under the System Management tab.
3. Click **New** for creating a new scheduled job. After that provide the job name as **Flat File** and in the Task field, select the value as **Flat File Connector User Reconciliation** from the lookup. Once the job is created, provide the values in the job as shown in Figure 6–15.

4. Add a scheduled task and add Flat File Connector User Reconciliation as the type as illustrated in Figure 6–15.

Figure 6–15 Scheduled Task Screen in Advanced Console



Event Management | Policies | System Management

Scheduler | Notification | System Configuration

Search Scheduled Jobs

Advanced Search

Actions ▾ View ▾

Job Name	Status
AD Group Delete Rec	Stopped
AD Group Lookup Rec	Stopped
AD Group Recon	Stopped
AD Organization Look	Stopped
AD Organization Rec	Stopped
AD User Target Delet	Stopped
AD User Target Recon	Stopped
AD User Trusted Dele	Stopped
AD User Trusted Rec	Stopped
Application Instance	Stopped
Attestation Grace Per	Stopped
Automated Retry of F	Stopped
Automatically Unlock	Stopped
Bulk Load Archival Jol	Stopped
Bulk Load Post Proces	Stopped
Catalog Synchronizat	Stopped
CRM On Demand Role	Stopped
CRM On Demand Use	Stopped
DataCollection Sched	Stopped
Delayed Delete User	Stopped
Disable/Delete User A	Stopped
Enable User After Sta	Stopped
Entitlement Assignme	Stopped
Entitlement List	Stopped
Entitlement Post Dele	Stopped
Evaluate User Policies	Stopped
Flat File	Stopped
Get SOD Check Resul	Stopped
Get SOD Check Resul	Stopped
Initiate Attestation P	Stopped
Issue Audit Messages	Stopped
Job History Archival	Stopped

Rows Selected 1

Welcome | Job Details

Job Details : Flat File

* Indicates required fields.

Apply | Run Now | Stop | Enable | Disable | Refresh

Job Information

Job Name: Flat File

Task: Flat File Connector User Reconciliation

* Start Date: June 12, 2012 2:58:41 AM (GMT-08:00) PST8PDT

* Retries: 2

Schedule Type: [✓ Periodic] Cron Single No pre-defined schedule

Job Periodic Settings

Run every: 11 days

Job Status

Current Status: Stopped

Last Run Start: June 11, 2012 3:25:35 AM PDT

Last Run End: June 11, 2012 3:25:38 AM PDT

Next Scheduled Run: June 12, 2012 2:58:41 AM PDT

Parameters

Filter:

Incremental Recon Date Attribute:

IT Resource Name: Flat File

Latest Token:

Object Type: User

Resource Object Name: Flat File

Job History

Actions ▾ View ▾ Show error details

Start Time	End Time	Job Status	Execution Status
June 11, 2012 3:25:	June 11, 2012 3:25:	Stopped	Success
June 11, 2012 3:22:	June 11, 2012 3:22:	Stopped	Success
June 11, 2012 3:16:	June 11, 2012 3:16:	Stopped	Success
June 11, 2012 3:08:	June 11, 2012 3:08:	Stopped	Success
June 11, 2012 3:04:	June 11, 2012 3:04:	Stopped	Success
June 11, 2012 3:02:	June 11, 2012 3:02:	Stopped	Success
June 11, 2012 3:00:	June 11, 2012 3:00:	Stopped	Success

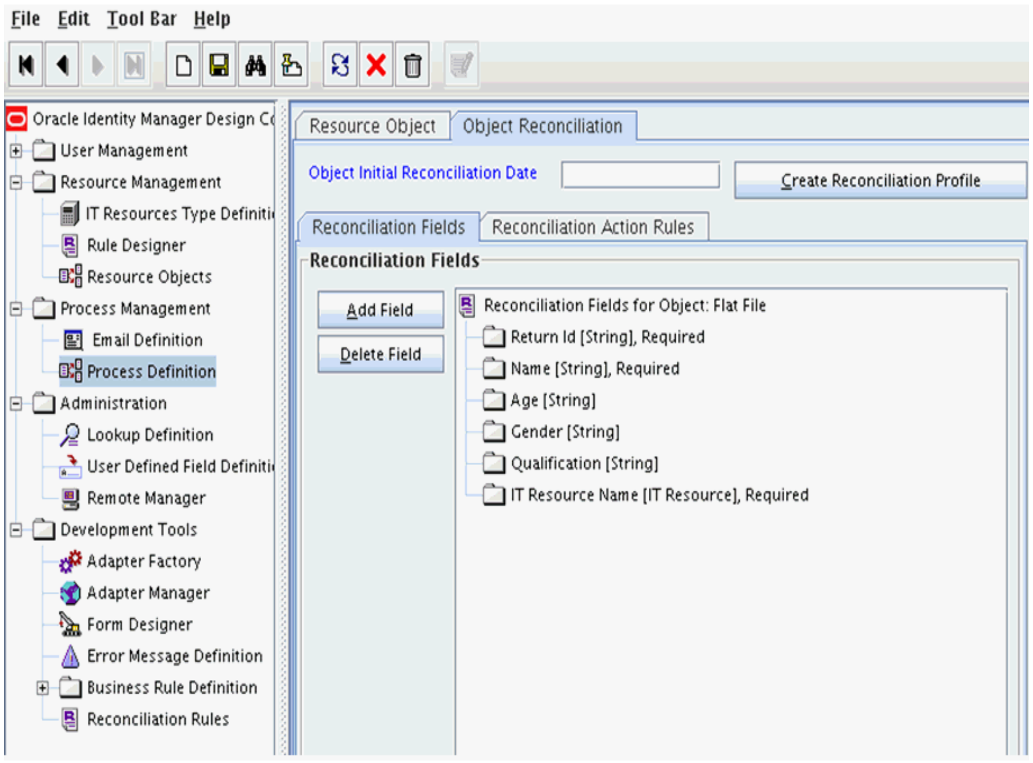
5. Set the parameters as follows:
   - IT Resource Name takes a value of Flat File.
   - Resource Object Name takes a value of FLATFILE.
   - Object Type takes a value of User.
6. Click **Apply**.

6.2.4.2 Creating a Reconciliation Profile

A reconciliation profile defines the structure of the object attributes while reconciliation. The reconciliation profile should contain all the attributes that have reconciliation support.

1. Log in to the Oracle Identity Manager Design Console.
2. Click **Resource Objects** under Resource Management.
3. Open the Flat File resource object.
4. Click the **Object Reconciliation** tab as illustrated in Figure 6–16.

Figure 6–16 Object Reconciliation in Design Console



Reconciliation Fields for Object: Flat File
- Return Id [String], Required
- Name [String], Required
- Age [String]
- Gender [String]
- Qualification [String]
- IT Resource Name [IT Resource], Required

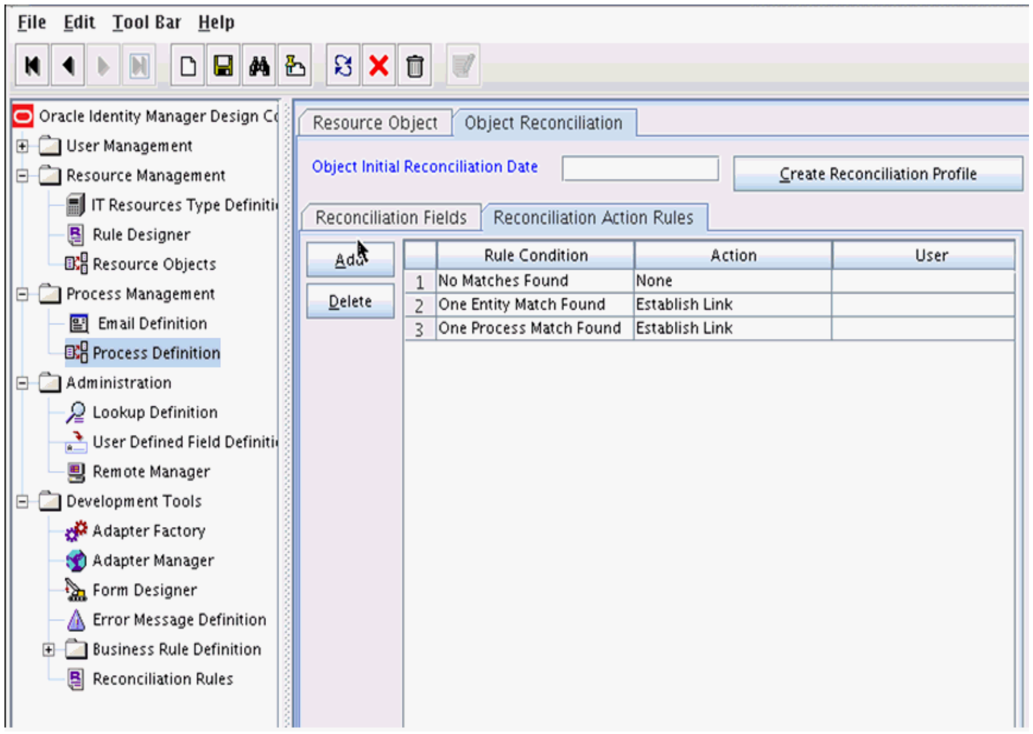
5. Add following reconciliation fields:
   - Return Id [String] , Required]
   - Name [String] , Required
   - Gender [String]
   - Age [String]
   - Gender [String]
   - IT Resource Name [IT Resource] , Required
6. Save the configuration.

6.2.4.3 Setting a Reconciliation Action Rule

A Reconciliation Action Rule defines the behavior of reconciliation. In this procedure, define the expected action when a match is found. This procedure assumes you are logged into the Oracle Identity Manager Design Console.

1. Open the Flat File resource object.
2. Click the **Object Reconciliation** tab.
3. Click the **Reconciliation Action Rules** tab in the right frame as illustrated in Figure 6–17.

Figure 6–17 Reconciliation Action Rules in Design Console



	Rule Condition	Action	User
1	No Matches Found	None	
2	One Entity Match Found	Establish Link	
3	One Process Match Found	Establish Link	

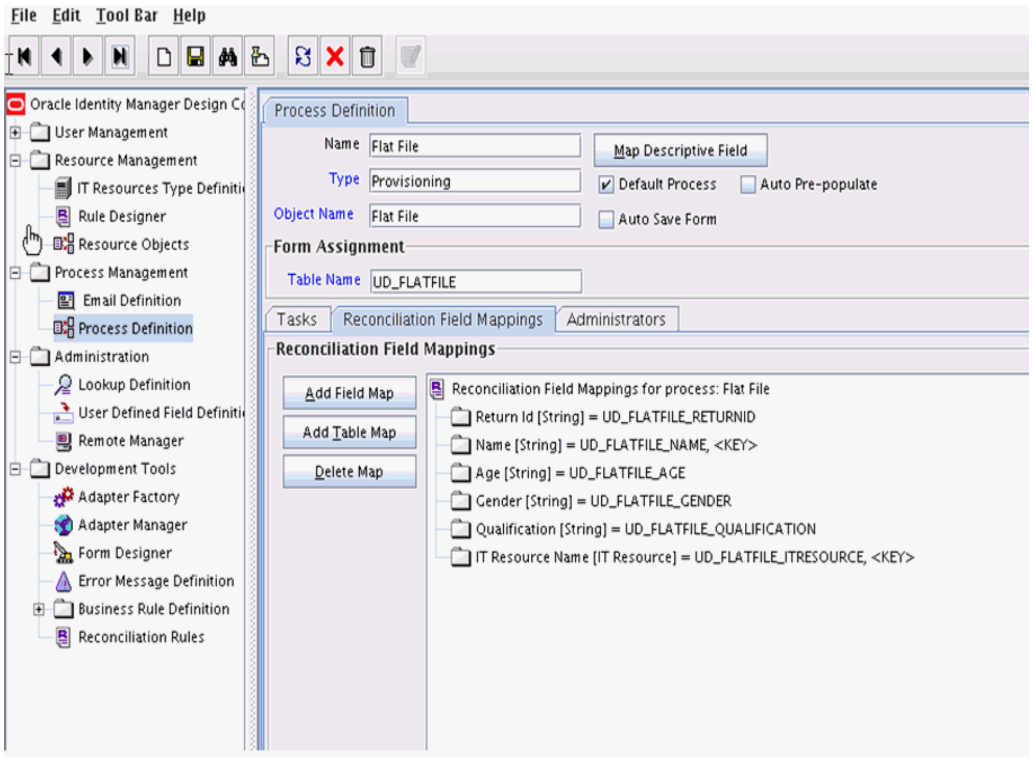
4. Add an action rule defined as One Process Match Found (Rule Condition) and Establish Link (Action).
5. Add an action rule defined as One Entity Match Found (Rule Condition) and Establish Link (Action).
6. Click **Create Reconciliation Profile**.
7. Click **Save**.

6.2.4.4 Creating Reconciliation Mapping

The reconciliation mapping has to be done in the process definition. This is to map the supported reconciliation fields (from resource object) to the process form fields. This mapping is needed only for configuring target reconciliation.

1. Log in to the Oracle Identity Manager Design Console.
2. Click **Process Definition** under Process Management.
3. Open the Flat File process definition.
4. Click the Reconciliation Field Mappings tab as illustrated in Figure 6–18.

Figure 6–18 Reconciliation Field Mapping in Design Console



Name: Flat File

Type: Provisioning

Object Name: Flat File

[✓] Default Process [ ] Auto Pre-populate

[ ] Auto Save Form

Form Assignment

Table Name: UD_FLATFILE

Reconciliation Field Mappings

Reconciliation Field Mappings for process: Flat File
- Return Id [String] = UD_FLATFILE_RETURNID
- Name [String] = UD_FLATFILE_NAME, <KEY>
- Age [String] = UD_FLATFILE_AGE
- Gender [String] = UD_FLATFILE_GENDER
- Qualification [String] = UD_FLATFILE_QUALIFICATION
- IT Resource Name [IT Resource] = UD_FLATFILE_ITRESOURCE, <KEY>

5. Add mappings between the reconciliation profile fields and the process form fields.
   - ReturnId[String] = UD_FLATFILE_RETURNID
   - Name[String] = UD_FLATFILE_NAME, <KEY>
   - Age[String] = UD_FLATFILE_AGE
   - Gender[String] = UD_FLATFILE_GENDER
   - Qualification[String] = UD_FLATFILE_QUALIFICATION
   - IT Resource Name[IT Resource] = UD_FLATFILE_ITRESOURCE,<KEY>
6. Save the configuration.

6.2.4.4.1 Field Flags Used in the Reconciliation Attributes Map 9

Note: These properties are advanced options and can be skipped for the current implementation of the connector

For reconciliation attributes mapping, the following field flags can be appended to the code key:

- **TRUSTED:** This must be specified in the Recon Attribute Map for the field that represents the status of the account. This flag must be specified only for trusted reconciliation. If this is specified, then the status of the account is either Active or Disabled. Otherwise, the status is either Enabled or Disabled. For example, the code key for a field with label Status whose value needs to be either Active/Disabled must look similar to Status[TRUSTED].

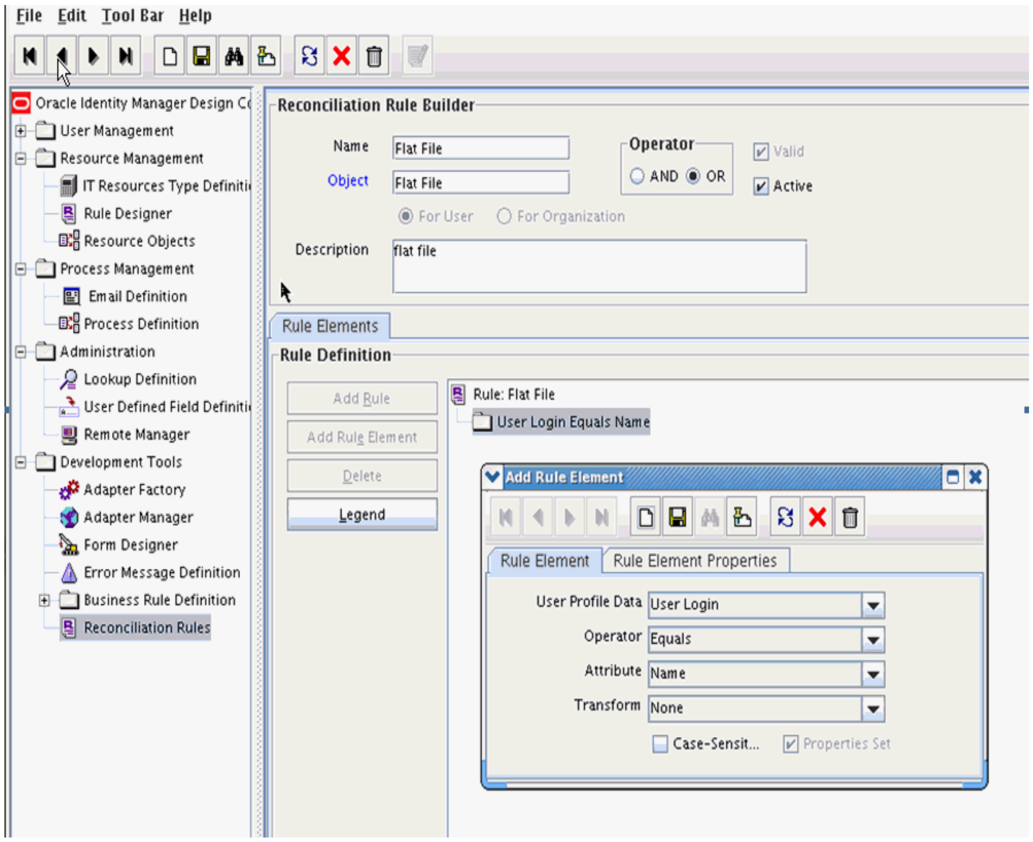
- **DATE:** In Recon Attribute Map, this must be specified for fields whose type need to be considered as Date. For example, the code key for a field with label Today whose value needs to be displayed in the date format must look similar to Today[DATE].

6.2.4.5 Defining a Reconciliation Matching Rule

A reconciliation matching rule defines the equation for calculating the user match.

1. Log in to the Oracle Identity Manager Design Console.
2. Open the **Reconciliation Rules** form under Development Tools.
3. Click **Add Rule**.

Figure 6–19 Adding Reconciliation Matching Rule



Reconciliation Rule Builder

Name: Flat File

Object: Flat File

Operator: AND [✓ OR]

[✓ Valid] [✓ Active]

[✓ For User] For Organization

Description: flat file

Rule: Flat File

User Login Equals Name

Add Rule Element

User Profile Data: User Login

Operator: Equals

Attribute: Name

Transform: None

Case-Sensit... [✓ Properties Set]

4. Select resource object Flat File.
5. Once the reconciliation rule element is added, make sure to check Active flag so that the reconciliation rule is made active.
6. Save and add the rule element.

   User Login from the user profile data equals the Name resource attribute.
7. Save the rule.

Note: You must recreate the reconciliation profile whenever you make any changes to the reconciliation rule.

6.3 Provisioning a Flat File Account

The flat file connector is ready to work so now the user needs to log in to Oracle Identity Manager and create an IT resource (target) using the following procedure.

- Create IT resource of type "Flat File".
- Provide the IT resource parameters as appropriate.
- Provide the configuration parameters in Lookup.FlatFile.Configuration as appropriate.

7 Integrating ICF with Oracle Identity Manager

Oracle Identity Manager's goal is to manage the business logic of Identity administration, and delegate the execution of provisioning and reconciliation operations to Identity Connector Framework (ICF). ICF with Oracle Identity Manager (OIM) unites all the scheduled tasks and the provisioning tasks for all ICF based connectors.

This chapter contains conceptual information about ICF-OIM integration in the sections:

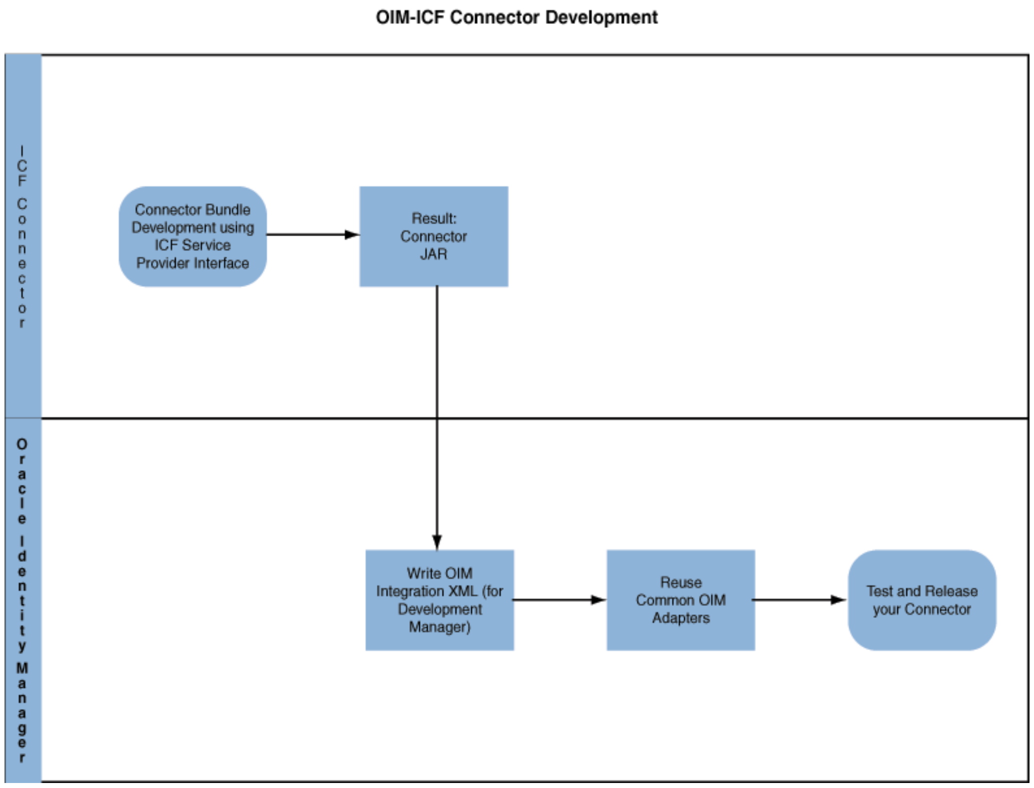
7.1 ICF Common

OIM ICF Integration Layer is an implementation of ICF API on one side and invokes OIM APIs (icf-oim-intg.jar) on the other side. This reduces the complexity of the connector developer as it provides API abstraction. It also support provisioning and reconciliation operations. See Section 7.5, "Provisioning" and Section 7.6, "Concepts of Reconciliation in ICF Common" for more information about provisioning and reconciliation using ICF Common.

7.2 Integration Architecture

Figure 7–1 is the ICF-OIM integration architecture.

Figure 7–1 OIM-ICF Connector Development Architecture


OIM-ICF Connector Development
ICF Connector
Oracle Identity Manager
Connector Bundle Development using ICF Service Provider Interface
Result: Connector JAR
Write OIM Integration XML (for Development Manager)
Reuse Common OIM Adapters
Test and Release your Connector

7.3 Global Oracle Identity Manager Lookups

Lookups is used to store Oracle Identity Manager configuration metadata. IT Resource parameter Configuration Lookup points to main Configuration Lookup that encapsulates all the Oracle Identity Manager specific configuration information.

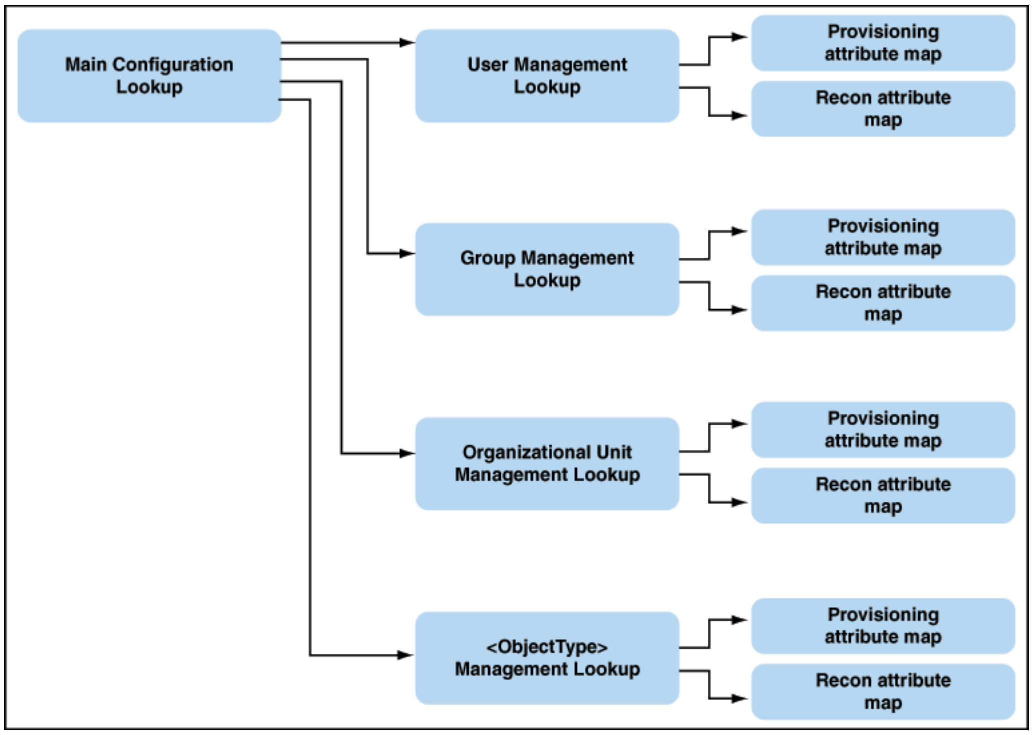
Based on the lookup configuration, you can classify your properties into the following three classes:

- IT Resource: connectivity properties: contains all properties that are used for making a connection to the target system.
- Main Configuration Lookup Configuration Properties: contains non-connectivity properties that alter the mode of reconciliation or provisioning, and are not required for connection. There is a thin line of difference between connectivity and configuration properties, therefore one property can be assigned to both of them.
- Object Type: specific lookups (for example, user management configuration), mapping lookups for specific object type (for example, User, Group, Organizational Unit).

Note: LOADFROMURL flag can be used in IT Resource or Main Configuration Lookup in the code (key) field, for example, sampleProperty[LOADFROMURL]. For properties marked as this, the value (decode value) is a URL. ICF integration will read the contents of the file stored in the given URL and use it as the value of given property at runtime. This is useful for large values that cannot fit directly into a lookup.

Figure 7–2 illustrates the global Oracle Identity Manager lookups from which most of the Connectors use the User Management Lookups.

Figure 7–2 Oracle Identity Manager Connector Lookup Hierarchy


Main Configuration Lookup
User Management Lookup
Provisioning attribute map
Recon attribute map
Group Management Lookup
Provisioning attribute map
Recon attribute map
Organizational Unit Management Lookup
Provisioning attribute map
Recon attribute map
<ObjectType> Management Lookup
Provisioning attribute map
Recon attribute map

This section discusses the following topics:

- Section 7.3.1, "Main Lookup Configuration"
- Section 7.3.2, "User Management Configuration"
- Section 7.3.3, "Recon Transformation Lookup (Lookup.CONNECTOR_NAME.UM.ReconTransformation)"
- Section 7.3.4, "Recon Validation Lookup (Lookup.CONNECTOR_NAME.UM.ReconValidation)"
- Section 7.3.5, "Optional Defaults Lookup"

7.3.1 Main Lookup Configuration

IT Resource parameter Configuration Lookup points to Main Configuration Lookup, which encapsulates all the Oracle Identity Manager specific configuration information.

Configuration lookup, denoted as Lookup.CONNECTOR_NAME.Configuration, is the top level entry that refers to subordinate lookups for reconciliation and provisioning. The configuration lookup has the structure shown in Table 7–1:

Table 7–1 Lookup Configuration for Connector

Configuration Key	Value	Description
Connector Name	org.identityconnectors.CONNECTOR_NAME.Connector	Identity Connector Main Class. This is the class that implements SPI operations of ICF framework.

Table 7–1 (Cont.) Lookup Configuration for Connector

Configuration Key	Value	Description
Bundle Name	org.identityconnectors.CONNECTOR_NAME	Identity Connector bundle name
Bundle Version	11.1.1.5.x	Identity Connector bundle version
User Configuration Lookup **Note**: Other object types may be defined, for example, for Generic LDAP connector: Group Configuration Lookup, OU Configuration Lookup.	Lookup.CONNECTOR_NAME.UM.Configuration	Link to User specific configuration lookup. **Note**: User should be the object type. If you need to support any other object type, you can use OBJECT_TYPE Configuration Lookup as the key.

7.3.2 User Management Configuration

These lookups control the mapping for provisioning and reconciliation. In addition, these lookups might also configure transformation and validation.

This lookup contains the following keys:

- Before Create Action Language: This key if present in the Lookup.CONNECTOR_NAME.UM.Configuration, which informs ICF that there is a script whose language is the value of this key. The value of this key (Groovy/cmd) informs the language of the script that needs to be executed by ICF before create operation.
- Before Create Action File: This key if present in the Lookup.CONNECTOR_NAME.UM.Configuration, informs ICF that a script represented by the value of this key needs to be executed by ICF before create action. This script must be accessible to Oracle Identity Manager Server.
- Before Create Action Target: This key if present in the Lookup.CONNECTOR_NAME.UM.Configuration, informs ICF that script as defined by previous two keys must be executed either on resource or on connector. Depending on this configuration the ICF API runScriptOnConnector or runScriptOnResource will be executed.

Table 7–2 describes the User Management lookup configuration.

Table 7–2 User Management Lookup Configuration for Connector

Configuration Key	Value	Mandatory Field Type	Description
Provisioning Attribute Map	Lookup.CONNECTOR_NAME.UM.ProvAttrMap	Y	This lookup contains the mapping between Oracle Identity Manager fields and identity connector attributes. The mapping is used during provisioning.

Table 7–2 (Cont.) User Management Lookup Configuration for Connector

Configuration Key	Value	Mandatory Field Type	Description
Recon Attribute Map	Lookup.CONNECTOR_NAME.UM.ReconAttrMap	Y	This lookup contains the mapping between Oracle Identity Manager reconciliation fields and identity connector attributes. The mapping is used during reconciliation.
Recon Attribute Defaults	Lookup.CONNECTOR_NAME.UM.ReconDefaults	N	This mapping contains the default values for Oracle Identity Manager attributes, that are substituted, if no value is provided by connector during reconciliation.
Recon Transformation Lookup	Lookup.CONNECTOR_NAME.UM.ReconTransformation	N	Lookup for Transformation by doing Reconciliation Task. Transformation is used in all Reconciliation Tasks except LookupReconTask.
Recon Validation Lookup	Lookup.CONNECTOR_NAME.UM.ReconValidation	N	Lookup used for Validation by running Reconciliation Task. Validation is used in all Reconciliation Tasks except LookupReconTask.
Recon Exclusion List	Lookup.CONNECTOR_NAME.UM.ReconExclusionList	N	Exclusion list is a way to address un-managed accounts for the connector. While reconciliation/provisioning. Any match from the exclusion list will not be processed by Oracle Identity Manager. There are two types of rules supported by the exclusion list: ■ Matching rules **Direct Matching Rule** Code Key: Reconciliation field name Decode Key: Excluded field value ■ Pattern Matching Rule Suffix with [PATTERN] tag to enable pattern matching Code Key: ReconFieldName[PATTERN] Decode Key: Exclusion pattern Exclusion patterns should follow the nomenclature defined in java.util.regex.Pattern See the Recon Exclusion List key in this table.
Provisioning Exclusion List	Lookup.CONNECTOR_NAME.UM.ProvExclusionList	N	In provisioning, code key is the Form label name, and decode key is the excluded value/pattern.

Table 7–2 (Cont.) User Management Lookup Configuration for Connector

Configuration Key	Value	Mandatory Field Type	Description
Provisioning Validation Lookup	Lookup.CONNECTOR_NAME.UM.ProvValidation	N	Lookup for Validation by Provisioning. ICF defines the concept of OperationOption, it is an extra parameter list, that can be sent to any operation. It is up to the connector implementation to define the use of these operation options.
Operation Options Map	Lookup.CONNECTOR_NAME.UM.OperationOptions	N	The code key is a constant Operation Options Map. The decode value name of lookup that will be used as a map of operation options. For example, in Lookup.Domino.UM.OperationOptions the code key is CACertifier[UPDATE,DELETE] and the decode value is CACertifier, which means that this attribute will be sent to calls of Update and Delete operations as an extra operation option. If you want to configure the action run, then you need to provide three parameters for scripting: ■ Language ■ File ■ Target
Scripting Attributes			The triggering time of the script is controlled by these labels in your lookup key: ■ Before ■ After The provisioning operation type that the script is attached on is controlled by these labels: ■ Create ■ Update ■ Delete
Before Create Action Language	SCRIPTING_LANGUAGE_NAME	N	Language of the Action which will be executed, for example, Groovy/cmd. If you want to configure the action run, then you need to provide three options, Language/File/Target You can configure Before/After actions for the following provisioning operations: Create/Update/Delete.

Table 7–2 (Cont.) User Management Lookup Configuration for Connector

Configuration Key	Value	Mandatory Field Type	Description
Before Create Action File	FILE_PATH	N	File containing script which needs to be executed. This file needs to be accessible to Oracle Identity Manager Server.
Before Create Action Target	Connector or Resource	N	Target of the action, can be Connector or Resource. Depending on this configuration the ICF API runScriptOnConnector or runScriptOnResource will be used.

7.3.3 Recon Transformation Lookup (Lookup.CONNECTOR_NAME.UM.ReconTransformation)

Transformation code is in an external Oracle Identity Manager Java Task, used in all Reconciliation Tasks except LookupReconTask. It is a Java class uploaded (transforming data coming from Target System during reconciliation) to Oracle Identity Manager repository.

The Java class performing transformation needs to have a method with the signature public Object transform(HashMap arg0, HashMap arg1, String arg2) implemented. ICF would look for this method with the exact signature.

Transform java class template is as follows:

```
public class MyTransformer implements
oracle.iam.connectors.common.transform.Transformation {
  public Object transform(java.util.HashMap hmUserDetails, java.util.HashMap
hmEntitlementDetails, String sField) {
    String sFirstName= (String)hmUserDetails.get("First Name");
    String sLastName= (String)hmUserDetails.get("Last Name");
    String sFullName=sFirstName+"."+sLastName;
    return sFullName;
  }
}
```

The name of lookup storing the Recon Transformation Lookup is defined in Main Configuration Lookup (Lookup.CONNECTOR_NAME.Configuration) as shown in Table 7–3.

Table 7–3 Reconciliation Transformation Lookup

Key	Value	Description
Recon Field Name	<transformationClassName> com.validationexample.MyTransform	Java class which performs transformation for this recon field.

7.3.4 Recon Validation Lookup (Lookup.CONNECTOR_NAME.UM.ReconValidation)

Validation code is in an external Oracle Identity Manager Java task, Used for validating data coming from Target System during Reconciliation. It is a Java class uploaded (transforming data coming from Target System during reconciliation) to Oracle Identity Manager repository.

The Java class performing validation needs to have a method with the signature public boolean validate (HashMap arg0, HashMap arg1, String arg2) implemented. ICF would look for this method with the exact signature.

The validation Java class template is as follows:

```
public class MyValidator implements
oracle.iam.connectors.common.validate.Validator {
  public boolean validate(java.util.HashMap hmUserDetails, java.util.HashMap
hmEntitlementDetails,String sField) throws
oracle.iam.connectors.common.ConnectorException {
    boolean isValid = false;
    // validation code goes HERE
    return isValid;
  }
}
```

The name of lookup storing the Recon Validation Lookup is defined in main configuration lookup (Lookup.CONNECTOR_NAME.Configuration) as shown in Table 7–4.

Table 7–4 Reconciliation Validation Lookup

Key	Value	Description
Recon Field Name	<transformationClassName> com.validationexample.MyValidator	Java class which performs validation for this recon field.

7.3.5 Optional Defaults Lookup

Missing values for reconciliation are substituted by default values defined in the following table. User Type is a required Oracle Identity Manager attribute, that typically is not contained on the target resource. You can set the default value in here.

For example, trusted reconciliation requires a set of attributes from the connector to have a non-empty value. However, not all resources can supply all of these attribute types, so you need to provide some default values. Table 7–5 lists all required attributes for reconciliation, and possible default values for them.

If connector can supply all attributes needed in reconciliation, then this table becomes optional.

Table 7–5 Lookup.CONNECTOR_NAME.UM.Recon.Defaults.Trusted Attriburtes

Key	Value
Last Name	CONNECTOR_DEPENDENT_VALUE
Organization	Xellerate users
User Type	End-User
Employee Type	Full-Time
First Name	CONNECTOR_DEPENDENT_VALUE

Note: These default values are supported only for single valued fields, which means the multivalued or child table attributes are not supported.

7.4 IT Resource

IT Resource contains connectivity parameters for Target System. These parameters are required for all the connectors using ICF integration.

Table 7–6 describes the common IT Resource parameters.

See Also: The documentation for the connector you are deploying for information about the IT Resource parameters of the target system and the Connector Server

Table 7–6 IT Resource Parameter

Parameter	Description
Connector Server Name	IT Resource name of Connector Server. The IT Resource needs to be of type Connector Server. This field is a mandatory field, but the value is optional.
Configuration Lookup	Name of the main configuration lookup. This field is a mandatory field

7.5 Provisioning

The section contains the following topics:

- Section 7.5.1, "ICF Provisioning Manager"
- Section 7.5.2, "Provisioning Lookup"
- Section 7.5.3, "Non-User Object Types"
- Section 7.5.4, "Optional Lookups for Provisioning"
- Section 7.5.5, "Optional Flags in Lookups for Provisioning Attribute Map"
- Section 7.5.6, "Compound attributes in Provisioning Attribute Map"

7.5.1 ICF Provisioning Manager

ICF Provisioning Manager unites the access to provisioning methods of connectors into one Java Task that serves all connectors.

The public methods are divided into four groups:

- Section 7.5.1.1, "APIs for Provisioning"
- Section 7.5.1.2, "Account Related Operations"
- Section 7.5.1.3, "Multivalued Operations"
- Section 7.5.1.4, "Other operations"

7.5.1.1 APIs for Provisioning

The following are the single-valued CRUD object types.

createObject

Creates object of a specified type on the target resource, the values are taken from the current Form.

Signature: public String createObject(String objectType)l

deleteObject

Deletes object of a specified type on the target resource.

Signature: public String deleteObject(String objectType)

updateAttributeValue

Updates object on target resource, only the attribute with the provided label is updated.

Signature: public String updateAttributeValue(String objectType, String attrFieldName)

updatePassword

Use this method in Adapter ONLY if you need to provide old password value, currently there is no way to get the old value using the formAPI. If you don't need old password value to change the password, use #updateAttributeValue(String, String) method instead.

Signature: public String updatePassword(String objectType, String pswdFieldLabel, String oldPassword)

7.5.1.2 Account Related Operations

The following are the account related provisioning operations.

enableUser

Deprecated, use enableObject() instead.

Signature: public String enableUser()

disableUser

Deprecated, use disableObject() instead.

Signature: public String disableUser()

enableObject

Example usage for User: enableObject("User").

Signature: public String enableObject(String objectType)

disableObject

Signature: public String disableObject(String objectType)

7.5.1.3 Multivalued Operations

The following are the multivalued operations used in provisioning.

updateAttributeValues

Use this method if there is a group update of fields. This will be useful when a set of attributes have to updated together.

Signature: public String updateAttributeValues(String objectType, String[] labels) public String updateAttributeValues(String objectType, Map<String, String> fields) public String updateAttributeValues(String objectType, Map<String, String> fields, Map<String, String> oldFields)}}}

addChildTableValue

Updates the target by adding the newly added row in child table.

Signature: public String addChildTableValue(String objectType, String childTableName, long childPrimaryKey)

removeChildTableValue

Updates the target by removing the row which was just deleted from child table.

Signature: public String removeChildTableValue(String objectType, String childTableName, Integer taskInstanceKey)

updateChildTableValue

Updates the target by removing the deleted row and adding the newly created row.

Signature: public String updateChildTableValue(String objectType, String childTableName, Integer taskInstanceKey, long childPrimaryKey)

updateChildTableValue

Updates values provided in child table on target resource.

Signature: public String updateChildTableValues(String objectType, String childTableName)

7.5.1.4 Other operations

The following is the other operation used in provisioning.

setEffectiveITResourceName

If the connector needs to use different IT Resource for provisioning operations, it can be set by this method.

Signature: public void setEffectiveITResourceName(String itResourceName)

7.5.2 Provisioning Lookup

Lookup.CONNECTOR_NAME.UM.ProvAttrMap contains basic attribute mapping for two classes of attributes:

- Single valued attributes: simple string key + value pairs.
- Multivalued attributes (Child tables in Oracle Identity Manager): These are further divided by the depth of hierarchy:
  - Simple multivalued attributes: represent records of data stored in child table, see second row in Table 7–7.
  - Complex multivalued attributes: multiple levels of embedded objects, see last row in Table 7–7.

Table 7–7 Provisioning Lookup Attributes

Key	Value	Description
Form Field Label	ConnectorAttributeName	This is a basic mapping type, simple Form Label Name to single value Connector Attribute Name

Table 7–7 (Cont.) Provisioning Lookup Attributes

Key	Value	Description
Child Form Name>~<Child Form Field Label	ConnectorAttributeName	This maps child form field to multivalued ConnectorAttributeName
Child Form Name>~<Child Form Field Label	ConnectorAttributeName>~<EmbeddedObjectClass>~<EmbeddedAttributeName	This maps child form field to EmbeddedAttribute of the embedded object, which object class is EmbeddedObjectClass and it is included in ConnectorAttributeName

7.5.3 Non-User Object Types

There are number of other entities that can be provisioned for example LDAP Organizational Unit (also called OU), or LDAP Group or Group. In this case you need to fill in the OBJECT_TYPE in the following examples:

Main Configuration Lookup Lookup.CONNECTOR_NAME.Configuration

Key	Value	Description
objectType Configuration Lookup	Lookup.<Connector Name>.<objectType>.ProvAttrMap	
Group Configuration Lookup	Lookup.LDAP.Group.ProvAttrMap	Example for LDAP Group

Provisioning Lookup Lookup.CONNECTOR_NAME.OBJECT_TYPE.ProvAttrMap

Key	Value	Description
FORM_FIELD_LABEL_ON_THE_PROCESS_FORM	Target system attribute name	Attribute mapping between Oracle Identity Manager and the connector.

7.5.4 Optional Lookups for Provisioning

Key	Value	Description
FORM_FIELD_NAME [Create, Update, Delete]	ConnectorOperationOptionName	This field is used for generic definition. For example, where the field is mapped to operation option for CreateOp that is sent to connector named as myOperationOption.
myField[Create]	myOperationOption	

7.5.4.1 Provisioning Validation Lookup

Validation code is in an external OIM Java Task, it is a Java class uploaded to Oracle Identity Manager repository. Validation java class template:

```
public class MyValidator implements
oracle.iam.connectors.common.validate.Validator {
  public boolean validate(java.util.HashMap hmUserDetails, java.util.HashMap
hmEntitlementDetails,String sField) throws
oracle.iam.connectors.common.ConnectorException {
    boolean isValid = false;
    // validation code goes HERE
    return isValid;
  }
}
```

The name of lookup storing the Recon Validation Lookup is defined in Main Configuration Lookup (Lookup.CONNECTOR_NAME.Configuration).

Key	Value	Description
Form Field Label	validatorClassName com.validationexamp le.MyValidator	Java class which performs validation for this recon field.

7.5.5 Optional Flags in Lookups for Provisioning Attribute Map

ICF-OIM Integration offers some advanced flags that modify the way provisioning is done. The following is the example for formats of flags in look up key:

```
<key value>[<flag>]
<key value>[<flag1, flag2, flag3>]
```

Let us assume you have a Group OIM attribute that is mapped to UnixGroup Connector attribute. This OIM attribute is populated by a UI lookup. The correct row in Provisioning lookup will be:

```
nullLookup key: Group[LOOKUP]
Lookup value: UnixGroup }}}
```

The following is the list of flags and their effects.

Provisioning Lookup Flag: TRUSTED

For some attributes (for example trusted reconciliation of __ENABLE__ attribute), you need to pass on different values for trusted and target mode of operation. For most of the connectors which support status Reconciliation use code key: Status[Trusted], and decode value: __ENABLE__.

Provisioning Lookup Flag: IGNORE

An attribute marked as IGNORE, will be ignored during provisioning.

Provisioning Lookup Flag: WRITEBACK

If a field has WRITEBACK property, then update of that form field is:

1. update the value on the target system
2. query the value back from the target system (in order to get a normalized value)
3. update this normalized value on the user form.

Provisioning Lookup Flag: DATE

Use this flag to mark date fields. Oracle Identity Manager will apply the localized date format to these fields.

Provisioning Lookup Flag: PROVIDEONPSWDCHANGE

Use this flag to mark additional attributes that are needed for password change operation. By default only __PASSWORD__ attribute is sent, if no flag is applied.

7.5.6 Compound attributes in Provisioning Attribute Map

ICF Common enables to use Groovy expressions on the right hand side, so that provisioned attribute can be computed based on multiple fields. For example, in Active Directory Connector, decode value for the name field is: .

```
__NAME__=CN=${Common_Name},${Organization_Name}
```

7.6 Concepts of Reconciliation in ICF Common

ICF Common leverages the definition and types of reconciliation defined by Oracle Identity Manager server. IT Resource Name / Resource Object Name and Object Type are mandatory attributes reconciliation using ICF Common. Any target system attribute can be used as Latest Token Attribute.

This section contains the following topics:

- Section 7.6.1, "Types of Reconciliation"
- Section 7.6.2, "List of Reconciliation Artifacts in Oracle Identity Manager"

7.6.1 Types of Reconciliation

Reconciliation involves pulling identities from resource (also referred as target) to destination (Oracle Identity Manager). Reconciliation can be classified based on following criteria:

- Destination type: trusted, target recon.
- Scope: full, incremental recon.

Table 7–8 illustrates the common reconciliation parameters.

Table 7–8 ICF Common Reconciliation Parameters

Parameter	Field Setting	Description
Filter	Optional	Filter to limit the number of reconciled accounts, or to select specific set of users.
IT Resource Name	Mandatory	Name of IT Resource instance to reconciliation.
Object Type	Constant	User object class
Resource Object Name	Constant	Determines what OIM Resource Object to use for reconciliation.

7.6.1.1 Target and Trusted Reconciliation

Scheduled task name include keywords such as trusted, target, to determine the type of destination. By choosing the scheduled task, it is determined whether trusted or target reconciliation is launched.

7.6.1.2 Full, Incremental Reconciliation

Full reconciliation involves reconciling all existing user records from the target system into Oracle Identity Manager. During Full Reconciliation, scheduled task is launched for the first time, it is run in full reconciliation mode and from next runs happen in incremental mode. It is possible to switch manually between full/incremental reconciliation modes by emptying the Latest Token field on the scheduled task.

If no value is supplied in Incremental Recon Date Attribute and Incremental Recon Attribute, reconciliation is considered as Target Recon.

The following scheduled tasks offer optional incremental reconciliation:

- Connector Target User Reconciliation
- Connector Trusted User Reconciliation

7.6.1.3 Advanced Incremental Reconciliation

The format of Latest Token is altered by setting the Recon Date Format scheduled task parameter. The formatting string needs to follow standard pattern used in Java. For more information about formatting string used in Java, see Java Doc on Oracle Technology Network.

By default the `Latest Token is` long value that holds Unix/POSIX time.

7.6.1.4 Delete Reconciliation

Some connectors supports both trusted and target reconciliation of deleted accounts. Target reconciliation evaluate which Oracle Identity Manager users have lost their account on the resource, and unassign this resource in Oracle Identity Manager. Trusted Delete Reconciliation goes further, and deletes the Oracle Identity Manager User.

7.6.1.5 Group Lookup Reconciliation

Some connectors may support reconciliation of Groups, or other object classes to Lookups.

Before the first use of provisioning with the connector, it is advised to launch Lookup reconciliation. This reconciliation populate the `Lookup.CONNECTOR_NAME.ObjectType` table with groups available on an IT Resource that is being reconciled. The reconciliation is performed by the Connector Lookup Reconciliation scheduled task.

You need to set the IT resource parameter name, the rest of the parameters are constant as shown in Table 7–8.

Table 7–9 illustrates the common reconciliation parameters.

Table 7–9 Common Group Lookup Parameters

Code Key	Decode Key	Object Type
Form field name	Connector attribute	Group, or other

For example, the list of names returned by the connector is used to populate the lookup for provisioning. When a new user is provisioned, the group field can display the list of available groups.

7.6.2 List of Reconciliation Artifacts in Oracle Identity Manager

In Oracle Identity Manager, there are two methods of control over reconciliation:

- Lookups for Reconciliation: they define mapping, transformation of the attributes.
- Scheduled tasks - they define the way reconciliation is executed on connector side, or determine account/lookup mode of reconciliation.

7.6.2.1 Lookups for Reconciliation

The following are the lookups for reconciliation:

Reconciliation Attribute Map Lookup

The reconciliation attribute map contains the following pairs:

- Code key: Resource Object reconciliation field name
- Decode: Target system attribute name

Table 7–10 illustrates this mapping (Lookup.CONNECTOR_NAME.UM.ReconAttrMap) used by Scheduled tasks that perform reconciliation.

> **Note:** Resource Objects are different for Trusted and Target mode of reconciliation.

Table 7–10 Attribute Mapping for Lookup.CONNECTOR_NAME.UM.ReconAttrMap

Key	Value	Description
Recon Field Name	ConnectorAttributeName	This is a basic mapping type, single value Connector Attribute Name to simple Recon Field.
Recon Field Name~Child Recon Field Name	ConnectorAttributeName	This maps multivalued ConnectorAttributeName to child recon field.
Recon Field Name~Child Recon Field Name	ConnectorAttributeName~EmbeddedObjectClass~EmbeddedAttribute	This maps embedded attribute to child recon field

Example showing Design Console updates to setup reconciliation with child table

The following is the example showing Design Console updates to setup reconciliation with child table:

- Child table name: UD_FF_CHILD
- Column name: UD_FF_CHILD_ROLE
- Field label: Role

  To set up reconciliation with the above child table:

  1. Open Resource Object under Resource Management.
  2. Create a new Reconciliation Data field under Object Reconciliation tab.

> **Note:** While creating a new Reconciliation Data field, you must ensure that the field name be Roles and Field Type be Multi-Valued Attribute. This represents the child table as a whole UD_FF_CHILD.

3. Right click on the newly created Reconciliation Data Field and define a new property field as **Role**. This represents the actual column of the child table UD_FF_CHILD_ROLE.
4. Open **Reconciliation Field Mapping** under Process Definition.
5. Click on **Add Table Map**.
6. Select Field Name as **Roles**.
7. Select Table Name as **UD_FF_CHILD**.
8. Right click on the newly created field name Roles, click on **Define proper field name**.
9. Select **Role** for field name.
10. Select Process data field as **UD_FF_CHILD_ROLE**.
11. Update Lookup.CONNECTOR_NAME.UM.ReconAttrMap to include new lookup field with code key = Roles~Role and decode = Role (this should be connector side attribute name).
12. Go back to Resource Object and create reconciliation profile.
13. Clear cache.

7.7 Predefined Scheduled Tasks

ICF-OIM integration provides the following list of predefined scheduled tasks that a connector supports:

- Section 7.7.1, "LookupReconTask"
- Section 7.7.2, "SearchReconTask"
- Section 7.7.3, "SearchReconDeleteTask"
- Section 7.7.4, "SyncReconTask"

7.7.1 LookupReconTask

This scheduled task is based on ICF SearchOp based reconciliation. Oracle Identity Manager form field of type lookup stores a set of predefined values. These values originate from the connector's search query. The Code Key Attribute is the form field's name, and the Decode Attribute is the name of attribute on the target system (also called Connector).

Internally, this task invokes a search operation on the connector for the given Object Type that is translated to ICF Object Class eventually.

Table 7–11 Identity Connector Lookup Reconciliation Attributes

Key	Value
IT Resource Name	Specifies the name of the IT resource for target system installation.
Object Type	User
Lookup Name	This attribute holds the name of the lookup definition that maps each lookup definition with the data source from which values must be fetched.
Decode Attribute	Specifies the Decode Key column of the lookup definition.

Table 7–11 (Cont.) Identity Connector Lookup Reconciliation Attributes

Key	Value
Code Key Attribute	Specifies the Code Key column of the lookup definition.
Filter	Allows to create sophisticated filtration expressions in order to speed up/refine scheduled task execution.

7.7.2 SearchReconTask

The following is the ICF SearchOp based reconciliation.

Table 7–12 Identity Connector Target Search Reconciliation Attributes

Key	Value
IT Resource Name	Specifies the name of the IT resource for target system installation.
Resource Object Name	Specifies the name of the Resource Object used for reconciliation.
Object Type	User
Filter	Allows to create sophisticated filtration expressions in order to speed up/refine scheduled task execution.
Latest Token	Used in Filter as one of the criteria in incremental reconciliation. Any target system attribute can be used as Latest Token Attribute. This value is calculated as follows: If a reconciliation has fetched 100 records and Timestamp is chosen as a Incremental Recon Attribute, then Latest Token = Max Timestamp of all 100 records. It is not the Schedule task execution end timestamp.
Incremental Recon Date Attribute (optional, type Date)	Attribute used to update Latest Token. **Note**: If no value is supplied in Incremental Recon Date Attribute, then reconciliation is considered as Target Reconciliation.
Incremental Recon Attribute (optional, type long)	Attribute used to update Latest Token. **Note**: If no value is supplied in Incremental Recon Attribute , then reconciliation is considered as Target Reconciliation.

7.7.3 SearchReconDeleteTask

This scheduled task is used for ICF SearchOp based reconciliation.

Table 7–13 Identity Connector Target Search Delete Reconciliation Attributes

Key	Value
IT Resource Name	Specifies the name of the IT resource for target system installation.
Resource Object Name	Specifies the name of the Resource Object used for reconciliation
Object Type	User
Filter	Allows to create sophisticated filtration expressions in order to speed up/refine scheduled task execution.

7.7.4 SyncReconTask

This scheduled task is used for `ICF SyncOp` based reconciliation. The Sync Token field persists the token of last synchronization.

Table 7–14 Identity Connector Target Sync Reconciliation Attributes

Key	Value
IT Resource Name	Specifies the name of the IT resource for target system installation.
Resource Object Name	Specifies the name of the Resource Object used for reconciliation
Object Type	User
Filter	Allows to create sophisticated filtration expressions in order to speed up/refine scheduled task execution.
Sync Token	Token of last synchronization.

7.8 ICF Filter Syntax

GroovyFilterBuilder allows to create sophisticated filtration expressions in order to speed up/refine scheduled task execution.

WARNING: The GroovyFilterBuilder uses the connector attribute name for filtration. See Connector documentation for the attribute name.

Examples

The following example could limit the number of reconciled accounts to only those, where account name starts with letter "a", this filter is denoted by the following expression:

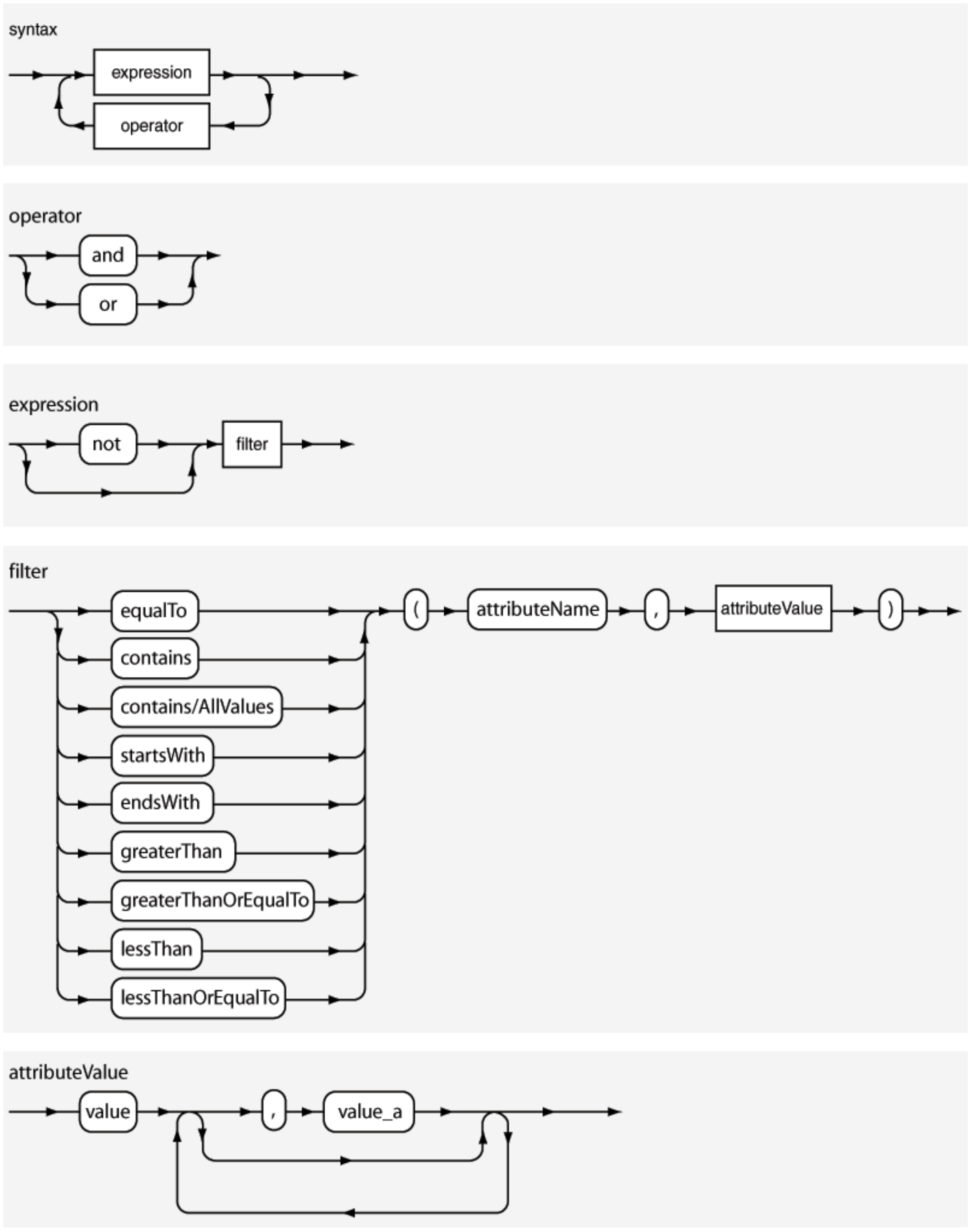
```
startsWith('__NAME__', 'a')
```

Some more advanced search could require to filter only those account names, which end with "z" letter, therefore the filter is:

```
startsWith('__NAME__', 'a') & endsWith('__NAME__', 'z')
```

Figure 7–3 shows the graphical scheme of Filter Syntax.

Figure 7–3 Graphical Representation of Filter Syntax



It is also possible to use a shortcut for and/or operators.

For example, <filter1> & <filter2> instead of and (<filter1>, <filter2>) , analogically replace or with |.

Definition in EBNF format:

The following is the Extended Backus–Naur Form (EBNF) description of the expression language used for Search Filters in reconciliation.

```
syntax = expression ( operator expression )*
operator = 'and' | 'or'
expression = ( 'not' )? filter
filter = ('equalTo' | 'contains' | 'containsAllValues' | 'startsWith' | 'endsWith'
| 'greaterThan' | 'greaterThanOrEqualTo' | 'lessThan' | 'lessThanOrEqualTo' )  '('
'attributeName' ',' attributeValue ')'
attributeValue = singleValue  |  multipleValues
singleValue = 'value'
multipleValues = '[' 'value_1' (',' 'value_n')* ']'
```

Table 7–15 lists the filter syntax that you can use and the corresponding description and sample values.

Note: Filters with wildcard characters are not supported.

Table 7–15 Keywords and Syntax for the Filter Attribute

Filter Syntax	Description
String Filters	
startsWith('*ATTRIBUTE_NAME*','*PREFIX*')	Records whose attribute value starts with the specified prefix are reconciled. **Example:** `startsWith('userPrincipalName','John')` In this example, all records whose userPrincipalName begins with 'John' are reconciled.
endsWith('*ATTRIBUTE_NAME*','*SUFFIX*')	Records whose attribute value ends with the specified suffix are reconciled. **Example:** `endsWith('sn','Doe')` In this example, all records whose last name ends with 'Doe' are reconciled.
contains('*ATTRIBUTE_NAME*','*STRING*')	Records where the specified string is contained in the attribute's value are reconciled. **Example:** `contains('displayName','Smith')` In this example, all records whose display name contains 'Smith' are reconciled.
containsAllValues('*ATTRIBUTE_NAME*',['*STRING1*','*STRING2*', . . . ,'*STRINGn*'])	Records that contain all the specified strings for a given attribute are reconciled. **Example:** `containsAllValues('objectClass',['person','top'])` In this example, all records whose objectClass contains both "top" and "person" are reconciled.
Equality and Inequality Filters	
equalTo('*ATTRIBUTE_NAME*','*VALUE*')	Records whose attribute value is equal to the value specified in the syntax are reconciled. **Example:** `equalTo('sAMAccountName','Sales Organization')` In this example, all records whose sAMAccountName is Sales Organization are reconciled.

Table 7–15 (Cont.) Keywords and Syntax for the Filter Attribute

Filter Syntax	Description
greaterThan('`ATTRIBUTE_NAME`','`VALUE`')	Records whose attribute value (string or numeric) is greater than (in lexicographical or numerical order) the value specified in the syntax are reconciled. **Example 1:** `greaterThan('cn','bob')` In this example, all records whose common name is present after the common name 'bob' in the lexicographical order (or alphabetical order) are reconciled. **Example 2:** `greaterThan('employeeNumber','1000')` In this example, all records whose employee number is greater than 1000 are reconciled.
greaterThanOrEqualTo('*ATTRIBUTE_NAME*','*VALUE*')	Records whose attribute value (string or number) is lexographically or numerically greater than or equal to the value specified in the syntax are reconciled. **Example 1:** `greaterThanOrEqualTo('sAMAccountName','S')` In this example, all records whose sAMAccountName is equal to 'S' or greater than 'S' in lexicographical order are reconciled. **Example 2:** `greaterThanOrEqualTo('employeeNumber','1000')` In this example, all records whose employee number is greater than or equal to 1000 are reconciled.
lessThan('*ATTRIBUTE_NAME*','*VALUE*')	Records whose attribute value (string or numeric) is less than (in lexicographical or numerical order) the value specified in the syntax are reconciled. **Example 1:** `lessThan('sn','Smith')` In this example, all records whose last name is present after the last name 'Smith' in the lexicographical order (or alphabetical order) are reconciled. **Example 2:** `lessThan('employeeNumber','1000')` In this example, all records whose employee number is less than 1000 are reconciled.
lessThanOrEqualTo('*ATTRIBUTE_NAME*','*VALUE*')	Records whose attribute value (string or numeric) is lexographically or numerically less than or equal to the value specified in the syntax are reconciled. **Example 1:** `lessThanOrEqualTo('sAMAccountName','A')` In this example, all records whose sAMAccountName is equal to 'A' or less than 'A' in lexicographical order are reconciled. **Example 2:** `lessThanOrEqualTo('employeeNumber','1000')` In this example, all records whose employee numer is less than or equal to 1000 are reconciled.
Complex Filters	

Table 7–15 (Cont.) Keywords and Syntax for the Filter Attribute

Filter Syntax	Description
<FILTER1> & *<FILTER2>*	Records that satisfy conditions in both filter1 and filter2 are reconciled. In this syntax, the logical operator & (ampersand symbol) is used to combine both filters. **Example:** `startsWith('cn', 'John') & endsWith('sn', 'Doe')` In this example, all records whose common name starts with John and last name ends with Doe are reconciled.
<FILTER1> | *<FILTER2>*	Records that satisfy either the condition in filter1 or filter2 are reconciled. In this syntax, the logical operator | (vertical bar) is used to combine both filters. **Example:** `contains('sAMAccountName', 'Andy') | contains('sn', 'Brown')` In this example, all records that contain 'Andy' in the sAMAccount Name attribute or records that contain 'Brown' in the last name are reconciled.
not(*<FILTER>*)	Records that do not satisfy the given filter condition are reconciled. **Example:** `not(contains('cn', 'Mark'))` In this example, all records that does not contain the common name 'Mark' are reconciled.

8 Using Java APIs for ICF Integration

To build a custom connector, you must first implement the Identity Connector (ICF-based connector) by implementing the ICF SPI. After this, you need to create Oracle Identity Manager artifacts to integrate the Identity Connector to Oracle Identity Manager, which can reuse ICProvisioningManager (part of ICF integration) in their Adapter Tasks to invoke provisioning operations on Identity Connector, and also can reuse Reconciliation Tasks that are implemented in ICF integration. Therefore, by using ICF integration, you need not write any integration code in java, ICF integration uses ICF APIs to access the Identity Connectors.

For information about Java APIs related to ICF integration, see *Oracle Fusion Middleware Java API Reference for Identity Connector Framework.*

For information about Java APIs related to Oracle Identity Manager, see *Oracle Fusion Middleware Java API Reference for Oracle Identity Manager*.

9

Configuring ICF Connectors

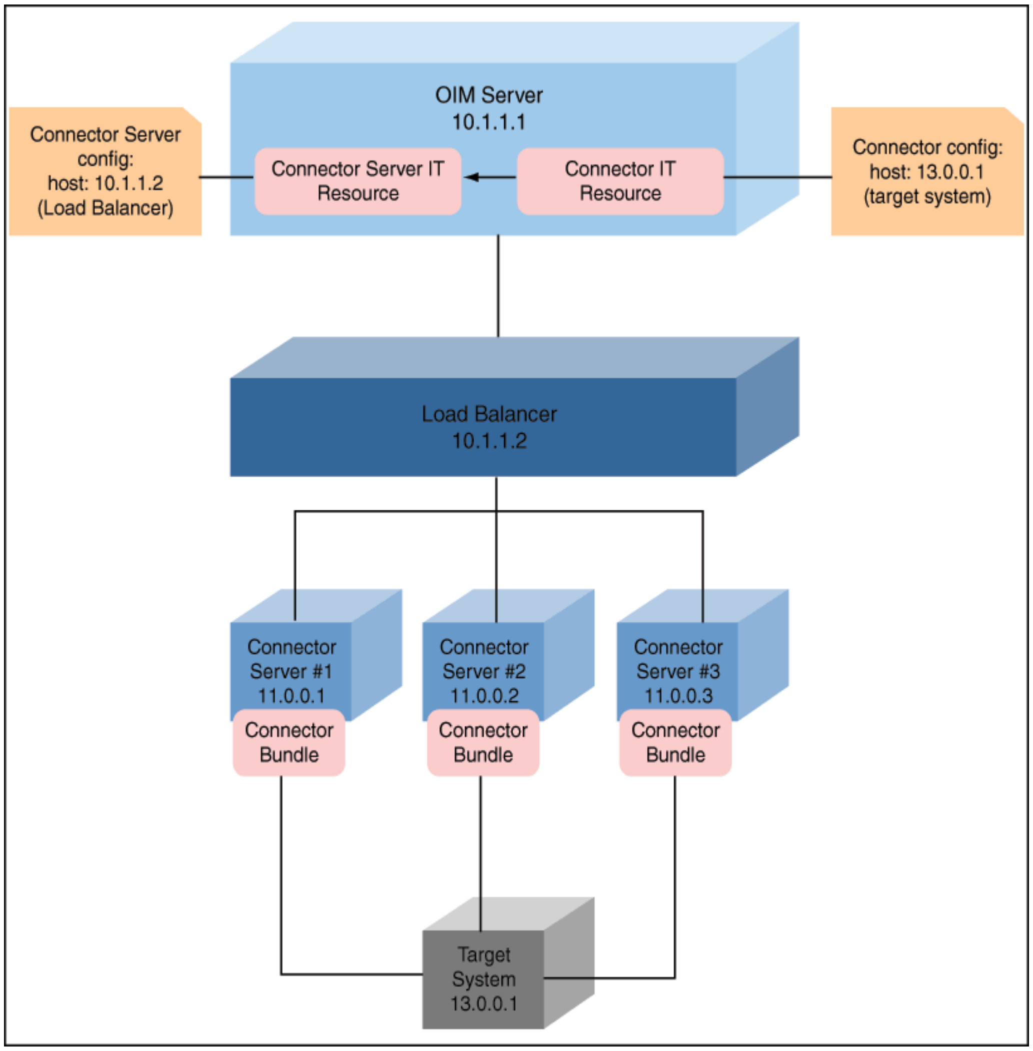
This chapter provides the information about the common customization procedures that needs to be performed for all ICF connectors.

The following are the topics discussed under this chapter:

9.1 Configuring Connector Load Balancer

A connector server is an application that enables remote execution of an Identity Connector. If there are multiple connector servers, then you must ensure the high availability of the connector server for the remote execution of the Identity connector and failover management. Therefore, you must configure a load balancer for a connector server. Figure 9–1 depicts the typical configuration for a cluster of connector servers. The flow in the figure is based on the assumption that the required connector bundle is deployed across all the connector servers.

Figure 9–1 Connector Server Load Balancer


Connector Server config: host: 10.1.1.2 (Load Balancer)
OIM Server 10.1.1.1
Connector Server IT Resource
Connector IT Resource
Connector config: host: 13.0.0.1 (target system)
Load Balancer 10.1.1.2
Connector Server #1 11.0.0.1
Connector Server #2 11.0.0.2
Connector Server #3 11.0.0.3
Connector Bundle
Connector Bundle
Connector Bundle
Target System 13.0.0.1

To configure the load balancer for a connector server:

1. Install connector server on nodes including the connector bundle. This involves copying and running the server binaries on all nodes.
2. Setup your load balancer so that every request on port 8759 (default for connector server, which is configurable) is being load balanced across the nodes created in Step 1.
3. Create a connector server IT resource, and point it to your host deployed with load balancer.
4. Configure your connector IT resource with the following details:
   - host: target address
   - connector server name: use the name created in Step 3.

Note: You must make sure to double-check that the incoming port of load balancer is same as the one given in connector server IT resource. In addition, you must check that the ports set up for cluster nodes match the one used for configuring your load balancer.

9.2 Configuring Validation of Data During Reconciliation and Provisioning

The Lookup.CONNECTOR_NAME.ProvValidations and Lookup.CONNECTOR_NAME.UM.ReconValidations lookup definitions hold single-valued data to be validated during provisioning and reconciliation operations, respectively.

For example, you can validate data fetched from the First Name attribute to ensure that it does not contain the number sign (#). In addition, you can validate data entered in the First Name field on the process form so that the number sign (#) is not sent to the target system during provisioning operations.

Note: The Lookup.CONNECTOR_NAME.UM.ProvValidations and Lookup.CONNECTOR_NAME.UM.ReconValidations lookup definitions are optional and do not exist by default.

You must add these lookups as decode values to the Lookup.CONNECTOR_NAME.UM.Configuration lookup definition to enable exclusions during provisioning and reconciliation operations. See the respective connector guide for more information about setting up the lookup definition for user operations.

To configure validation of data:

1. Write code that implements the required validation logic in a Java class with a fully qualified domain name (FQDN), such as `org.identityconnectors.CONNECTOR_NAME.extension.CONNECTOR_NAMEValidator`.

   This validation class must implement the validate method. The following sample validation class checks if the value in the First Name attribute contains the number sign (#):

```
package com.validationexample;

import java.util.HashMap;

public class MyValidator {
    public boolean validate(HashMap hmUserDetails, HashMap
hmEntitlementDetails, String sField) throws ConnectorException {

        /* You must write code to validate attributes. Parent
                 * data values can be fetched by using hmUserDetails.get(field)
                 * For child data values, loop through the
                 * ArrayList/Vector fetched by hmEntitlementDetails.get("Child
Table")
                 * Depending on the outcome of the validation operation,
                 * the code must return true or false.
                 */
        /*
        * In this sample code, the value "false" is returned if the field
        * contains the number sign (#). Otherwise, the value "true" is
        * returned.
        */
        boolean valid = true;
        String sFirstName = (String) hmUserDetails.get(sField);
        for (int i = 0; i < sFirstName.length(); i++) {
            if (sFirstName.charAt(i) == '#') {
                valid = false;
                break;
            }
        }
        return valid;

    }
}
```

2. Log in to the Design Console.
3. Create one of the following new lookup definitions:
   - To configure validation of data for reconciliation:

     `Lookup.CONNECTOR_NAME.UM.ReconValidations`
   - To configure validation of data for provisioning:

     `Lookup.CONNECTOR_NAME.UM.ProvValidations`
4. In the **Code Key** column, enter the resource object field name that you want to validate. For example, `Alias`.
5. In the **Decode** column, enter the class name. For example, `org.identityconnectors.CONNECTOR_NAME.extension.CONNECTOR_NAMEValidator`.
6. Save the changes to the lookup definition.
7. Search for and open the **Lookup.CONNECTOR_NAME.UM.Configuration** lookup definition.
8. In the **Code Key** column, enter one of the following entries:
   - To configure validation of data for reconciliation:

     `Recon Validation Lookup`
   - To configure validation of data for provisioning:

     `Provisioning Validation Lookup`
9. In the **Decode** column, enter one of the following entries:
   - To configure validation of data for reconciliation:

     `Lookup.CONNECTOR_NAME.UM.ReconValidations`
   - To configure validation of data for provisioning:

     `Lookup.CONNECTOR_NAME.UM.ProvValidations`
10. Save the changes to the lookup definition.
11. Create a JAR with the class and upload it to the Oracle Identity Manager database as follows:

    Run the Oracle Identity Manager Upload JARs utility to post the JAR file created in Step 7 to the Oracle Identity Manager database. This utility is copied into the following location when you install Oracle Identity Manager:

    Note: Before you use this utility, verify that the `WL_HOME` environment variable is set to the directory in which Oracle WebLogic Server is installed.

For Microsoft Windows:

OIM_HOME/server/bin/UploadJars.bat

For UNIX:

OIM_HOME/server/bin/UploadJars.sh

When you run the utility, you are prompted to enter the login credentials of the Oracle Identity Manager administrator, URL of the Oracle Identity Manager host computer, context factory value, type of JAR file being uploaded, and the location from which the JAR file is to be uploaded. Select 1 as the value of the JAR type.

> **See Also:** "Migrating JARs and Resource Bundle" on page 24-4 for detailed information about the Upload JARs utility

12. Run the PurgeCache utility to clear content related to request datasets from the server cache.
13. Perform reconciliation or provisioning to verify validation for the field, for example, Alias.

9.3 Configuring Transformation of Data During User Reconciliation

The Lookup.CONNECTOR_NAME.UM.ReconTransformations lookup definition holds single-valued user data to be transformed during reconciliation operations. For example, you can use First Name and Last Name values to create a value for the Full Name field in Oracle Identity Manager.

> **Note:** The Lookup.CONNECTOR_NAME.UM.ReconTransformations lookup definition is optional and does not exist by default.
>
> You must add this lookup as decode value to the Lookup.CONNECTOR_NAME.UM.Configuration lookup definition to enable exclusions during provisioning and reconciliation operations. See the respective connector guide for more information about setting up the lookup definition for user operations.

To configure transformation of single-valued user data fetched during reconciliation:

1. Write code that implements the required transformation logic in a Java class with a fully qualified domain name (FQDN), such as `org.identityconnectors.CONNECTOR_NAME.extension.CONNECTOR_NAMETransformation`.

   This transformation class must implement the transform method. The following sample transformation class creates a value for the Full Name attribute by using values fetched from the First Name and Last Name attributes of the target system:

```
package com.transformationexample;

import java.util.HashMap;


public class MyTransformer {
    public Object transform(HashMap hmUserDetails, HashMap
hmEntitlementDetails, String sField) throws ConnectorException {
        /*
        * You must write code to transform the attributes.
        * Parent data attribute values can be fetched by
        * using hmUserDetails.get("Field Name").
        * To fetch child data values, loop through the
        * ArrayList/Vector fetched by hmEntitlementDetails.get("Child
Table")
        * Return the transformed attribute.
        */
        String sFirstName = (String) hmUserDetails.get("First Name");
        String sLastName = (String) hmUserDetails.get("Last Name");
        return sFirstName + "." + sLastName;

    }
}
```

2. Log in to the Design Console.
3. Create a new lookup definition, **Lookup.CONNECTOR_NAME.UM.ReconTransformations.**
4. In the **Code Key** column, enter the resource object field name you want to transform. For example, `Alias`.
5. In the **Decode** column, enter the class name. For example, `org.identityconnectors.CONNECTOR_NAME.extension.CONNECTOR_NAMETransformation`.
6. Save the changes to the lookup definition.
7. Search for and open the **Lookup.CONNECTOR_NAME.UM.Configuration** lookup definition.
8. In the **Code Key** column, enter `Recon Transformation Lookup`.
9. In the **Decode** column, enter `Lookup.CONNECTOR_NAME.UM.ReconTransformations`.
10. Save the changes to the lookup definition.
11. Create a JAR with the class and upload it to the Oracle Identity Manager database as follows:

    Run the Oracle Identity Manager Upload JARs utility to post the JAR file created in Step 7 to the Oracle Identity Manager database. This utility is copied into the following location when you install Oracle Identity Manager:

    Note: Before you use this utility, verify that the `WL_HOME` environment variable is set to the directory in which Oracle WebLogic Server is installed.

    For Microsoft Windows:

    OIM_HOME/server/bin/UploadJars.bat

    For UNIX:

    OIM_HOME/server/bin/UploadJars.sh

    When you run the utility, you are prompted to enter the login credentials of the Oracle Identity Manager administrator, URL of the Oracle Identity Manager host computer, context factory value, type of JAR file being uploaded, and the location from which the JAR file is to be uploaded. Select 1 as the value of the JAR type.

See Also: "Migrating JARs and Resource Bundle" on page 24-4 for detailed information about this utility.

12. Run the PurgeCache utility to clear content related to request datasets from the server cache.
13. Perform reconciliation to verify transformation of the field, for example, Alias.

9.4 Configuring Resource Exclusion Lists

The Lookup.CONNECTOR_NAME.UM.ProvExclusionList and Lookup.CONNECTOR_NAME.UM.ReconExclusionList lookup definitions hold user IDs of target system accounts for which you do not want to perform provisioning and reconciliation operations, respectively.

Note: The Lookup.CONNECTOR_NAME.UM.ProvExclusionList and Lookup.CONNECTOR_NAME.UM.ReconExclusionList lookup definitions are optional and do not exist by default.

You must add these lookups as decode values to the Lookup.CONNECTOR_NAME.UM.Configuration lookup definition to enable exclusions during provisioning and reconciliation operations. See the respective connector guide for more information about setting up the lookup definition for user operations.

The following is the format of the values stored in these lookups:

Code Key	Decode	Sample Values
User Login Id resource object field name	User ID of a user	Code Key: User Login Id Decode: User001
User Login Id resource object field name with the [PATTERN] suffix	A regular expression supported by the representation in the `java.util.regex.Pattern` class	Code Key: User Login Id[PATTERN] To exclude users matching any of the user ID 's User001, User002, User088, then: Decode: User001 | User002 | User088 To exclude users whose user ID 's start with 00012, then: Decode: 00012* **See Also:** For information about the supported patterns, visit `http://download.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html`

To add entries in the lookup for exclusions during provisioning operations:

1. On the Design Console, expand **Administration** and then double-click **Lookup Definition.**
2. Create a new lookup definition, **Lookup.CONNECTOR_NAME.UM.ProvExclusionList.**

Note: To specify user IDs to be excluded during reconciliation operations, create a new lookup definition called Lookup.CONNECTOR_NAME.UM.ReconExclusionList and add entries to that lookup.

---

3. Click **Add.**
4. In the Code Key and Decode columns, enter the first user ID to exclude.

Note: The Code Key represents the resource object field name on which the exclusion list is applied during provisioning operations.

---

5. Repeat Steps 3 and 4 for the remaining user IDs to exclude.

   For example, if you do not want to provision users with user IDs User001, User002, and User088 then you must populate the lookup definition with the following values:

Code Key	Decode
User Login Id	User001
User Login Id	User002
User Login Id	User088

   You can also perform pattern matching to exclude user accounts. You can specify regular expressions supported by the representation in the `java.util.regex.Pattern` class.

   See Also: For information about the supported patterns, visit `http://download.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html`

   For example, if you do not want to provision users matching any of the user IDs User001, User002, and User088, then you must populate the lookup definition with the following values:

Code Key	Decode
User Login Id[PATTERN]	User001 | User002 | User088

   If you do not want to provision users whose user IDs start with 00012, then you must populate the lookup definition with the following values:

Code Key	Decode
User Login Id[PATTERN]	00012*

6. Click **Save**.

9.5 Setting SSL for Connector Server and OIM

To set up the SSL communication between Connector Server and Oracle Identity Manager:

1. Generate a new SSL key (or you can reuse your existing key):

   ```
   keytool -genkey -alias keyconnserv -keyalg dsa -keystore <yourKeyStore.jks>
   -storepass <yourPassword> -validity 360
   ```

2. Export the newly generated public key:

   ```
   keytool -export -keystore <yourKeyStore.jks> -storepass <yourPassword> -alias
   keyconnserv -file icfkey-public.cer
   ```

3. Configure your Connector Server for SSL, and start using the new keystore set in Step 1.
4. Import the public key generated in Step 2 (icfkey-public.cer) to OIM keystore.
5. Configure IT Resource such as host, port, and so on. These parameters will be passed on to Connector Server (an extra field of IT Resource).
6. Configure Connector Server, using SSL:
   a. Deploy an SSL certificate to the Connector Server's system.
   b. Configure your Connector Server to provide SSL sockets.
   c. Configure your application to communicate with the Connector Server using SSL.

      Refer to the target system's manual for specific notes on configuring connections to identity connector servers. You will indicate to your application that an SSL connection is required when establishing a connection for each SSL-enabled connector server. Additionally, if any of the SSL certificates used by your connector servers are issued by a non-standard certificate authority, your application must be configured to respect the additional authorities. Refer to your manual for notes regarding certificate authorities.

      Note:

      Java applications may solve the issue of non-standard certificate authorities by expecting the following Java system properties to be passed when launching the application:

      - javax.net.ssl.trustStorePassword

        For example:

        -Djavax.net.ssl.trustStorePassword=changeit

      - javax.net.ssl.trustStore

        For example:

        -Djavax.net.ssl.trustStore=/usr/myApp_cacerts

      Alternately, the non-standard certificate authorities may be imported to the standard ${JAVA_HOME}/lib/security/cacerts directory.

7. Import the public key generated in Step 2 to OIM keystore.

   If you follow to choose the default Weblogic keystore, perform the following:

```
keytool -import -trustcacerts -alias icfkey -file icfkey-public.cer -keystore
<pathToYouKeystore>
```

For example default Weblogic keystores are: server/lib/DemoTrust.jks and server/lib/DemoIdentity.jks.

9.5.1 Troubleshooting SSL

The following is an example of exception in connector server logs:

java.net.SocketException: Default SSL context init failed: null

This means that the path to keystore is incorrect. To handle this exception, make sure you provide the following full/absolute path:

For UNIX

./connectorserver.sh /run "-J-Djavax.net.ssl.keyStore=/path/to/mykeystore.jks" "-J-Djavax.net.ssl.keyStorePassword=changeit"

For Windows

./connectorserver.sh /run "-J-Djavax.net.ssl.keyStore=C:\path\to\mykeystore.jks" "-J-Djavax.net.ssl.keyStorePassword=changeit"

You must also ensure the following check points:

- Check your configuration folder for the setting of connector server configuration to use SSL
- Restart your WLS after importing public keys from the connector server, if the public key present in OIM keystore

9.6 Adding Target System Attributes

Adding target system attributes includes the following subsections:

- Section 9.6.1, "Adding Target System Attributes for Provisioning"
- Section 9.6.2, "Adding Target System Attributes for Target Reconciliation"
- Section 9.6.3, "Adding Target System Attributes for Trusted Reconciliation"

Note: If you add an attribute with a Date type field, make sure that you add the `[Date]` suffix in the Lookup definition code key.

For example, if you add _LAST_PASSWORD_CHANGE_DATE_, when you make changes in the code key for Lookup.CONNECTOR_NAME.UM.ReconAttrMap or Lookup.CONNECTOR_NAME.UM.ProvAttrMap, specify the attribute as:

```
_LAST_PASSWORD_CHANGE_DATE_[Date]
```

9.6.1 Adding Target System Attributes for Provisioning

By default, the target system attributes are mapped for provisioning between Oracle Identity Manager and the target system. If required, you can map additional attributes for provisioning by performing these steps.

Note: In this section, the term "attribute" refers to the identity data fields that store user data.

To add a target system attribute for provisioning, follow these steps:

1. Add a new form field. To add a new field to the Process form:

   a. Open the Form Designer form. This form is in the Development Tools folder of the Oracle Identity Manager Design Console.

   b. Query for the UD_CONNECTOR_NAMECON form.

   c. Click **Create New Version**. The Create a New Version dialog box is displayed.

   d. In the Label field, enter the name of the version.

   e. Click **Save** and close the dialog box.

   f. From the Current Version box, select the version name that you entered in the Label field in Step 4.

   g. On the Additional Columns tab, click **Add**.

   h. Specify the new field name and other values.

   i. Click **Save**.

   j. Click **Make Version Active** to make the new form field visible to the user.

   Now, if you go to Oracle Identity Manager and try to provision a new user to Connector, you should see the new form field. Next, you must add the new form field to the Provisioning Mapping Lookup.

2. Add the new field to the Provisioning Mapping Lookup. After creating a new form field, you must add that field to the Provisioning Mapping Lookup, as follows:

   a. Expand **Administration** and then double-click **Lookup Definition**.

   b. In the Lookup Definition window, search for CONNECTOR_NAME.

   The Design Console returns Lookup.CONNECTOR_NAME.UM.ProvAttrMap.

   c. Select the Lookup Definition Table tab, and select **Lookup.CONNECTOR_NAME.UM.ProvAttrMap**.

   The Lookup Code Information tab maps the Oracle Identity Manager form field names and the CONNECTOR_NAME Identity Connector attributes. Where the Code Key column contains the Oracle Identity Manager field labels and the Decode column contains the attribute names supported by the CONNECTOR_NAME identity connector.

   d. Add a new record for the new form field. Type the new form field name into the Code Key column and type the CONNECTOR_NAME identity connector attribute name into the Decode column.

   e. Click **Save**.

   Now, when you create a new CONNECTOR_NAME user, the connector will get the new attribute as part of the create operation.

   At this point, the process task only handles creates. Next, you must change the process task to also handle updates. Instructions are described in the next steps.

3. Change the process task to handle updates, as follows:

   a. In the Design Console, expand **Process Management** and then double-click **Process definition**.

   b. Search for and select process CONNECTOR_NAME User.

   c. In the Task column, look for an update task that is similar to the one you want to add and select that entry.

   d. Click **Add**.

   e. In the Creating New Task dialog, select the General tab and enter a Task Name and a Task Description.

      The Task Name is important because it will be the form name field. Be sure to include the event you want the task to handle. For example, if you add the Building field for provisioning, then add the Building Updated task. Now, this update event will be triggered when the Building field is updated.

   f. In the Task Properties section, set the following properties as noted:

      -Conditional: Enabled

      -Required for Completion: Disabled

      -Disable Manual Insert: Disabled

      -Allow Cancellation while Pending: Enabled

      -Allow Multiple Instances: Enabled

      You do not have to change any of the remaining properties.

   g. Save your changes.

   h. To add an Event Handler, select the Integration tab, and then click **Add**.

   i. When the Handler Select dialog box displays, select Adapter as the handler type and then perform the following steps:

      Select adapter **adpCONNECTOR_NAMECONNECTORUPDATEATTRIBUTEVALUE** and click **Save**.

      Map all of the variables that are configured for the event adapter.

      In the Adapter Variables section, double-click a variable name to open the Edit Data Mapping For Variable dialog box. Specify the following values for each variable in turn. Be sure to save your changes after each mapping.

Variable Name	Map To	Qualifier	Literal Value
itResourceFieldName	Literal	String	UD_CONNECTOR_NAMECON_SERVER
processInstanceKey	Process Data	Process Instance	
Adapter return value	Response Code		
objectType	Literal	String	User
attrName	Literal	String	Enter your new label

   j. Save and close the Creating New Task dialog.

k. Check the Task column on the Process Definition tab to verify that the new process task is listed. Also verify that the new form field is available and working in Oracle Identity Manager.

9.6.2 Adding Target System Attributes for Target Reconciliation

By default, the target system attributes are mapped for reconciliation between Oracle Identity Manager and the target system. If required, you can map additional attributes for target reconciliation as described in this section.

Note:

- Perform this procedure only if you want to add new target system attributes for reconciliation.
- In the following steps, a new attribute called BUILDING will be added, its connector attribute name is BUILDING, and the form field name is Building. Names are case-sensitive.

To add a new target system attribute for target reconciliation, follow these steps:

1. In the resource object definition, add a reconciliation field corresponding to the new attribute, as follows:
   a. Open the Resource Objects form. This form is in the Resource Management folder.
   b. Click **Query for Records**.
   c. On the Resource Objects Table tab, double-click the **CONNECTOR_NAME User** resource object to open it for editing.
   d. On the Object Reconciliation tab, click **Add Field** to open the Add Reconciliation Field dialog box.
   e. Specify a value for the field name that is the name of the new Attribute on your Form.

      For example: Building
   f. From the Field Type list, select a data type for the field.

      For example: String
   g. Save the values that you enter, and then close the dialog box.
   h. If required, repeat Steps d through g to map more fields.
   i. Click **Create Reconciliation Profile**. This copies changes made to the resource object into the MDS.
2. If a corresponding field does not exist in the process form, then add a new column in the process form, as follows:
   a. Open the Form Designer form. This form is in the Development tools folder.
   b. Query for the UD_CONNECTOR_NAMECON form.
   c. Click **Create New Version**. The Create a New Version dialog box is displayed.
   d. In the Label field, enter the name of the version.
   e. Click **Save** and close the dialog box.

f. From the Current Version box, select the version name that you entered in the Label field in Step 3.

g. On the Additional Columns tab, click **Add**.

h. In the Name field, enter the name of the data field and then enter the other details of the field.

Note: Repeat Steps g and h if you want to add more attributes.

i. Click Save and then click Make Version Active.

3. Modify the process definition to include the mapping between the newly added attribute and the corresponding reconciliation field:

a. Open the Process Definition form. This form is in the Process Management folder of the Design Console.

b. Click the **Query for Records** icon.

c. On the Process Definition Table tab, double-click the **CONNECTOR_NAME User** process definition.

d. On the Reconciliation Field Mappings tab, click **Add Field Map** to open the Add Reconciliation Field Mapping dialog box.

e. From the Field Name list, select the name of the resource object that you added in Step 2e.

f. Double-click Process Data Field and select the corresponding process form field from the Lookup dialog box. Then, click **OK**.

g. Click **Save** and close the dialog box.

h. If required, repeat Steps c through g to map more fields.

4. Go to the reconciliation lookup, Lookup.CONNECTOR_NAME.UM.ReconAttrMap, and add a new record for the new attribute using the following values:

- Code Key - Name of the reconciliation field
- Decode - Name of the CONNECTOR_NAME attribute

5. In the Design Console, regenerate the reconciliation profile for the Resource Object.

9.6.3 Adding Target System Attributes for Trusted Reconciliation

By default, the attributes for trusted source reconciliation are mapped between Oracle Identity Manager and the target system. If required, you can map additional attributes for trusted reconciliation as described in this section.

Note:

- Perform this procedure only if you want to add new target system attributes for reconciliation.
- In the following steps, a new attribute called BUILDING will be added, its connector attribute name is BUILDING, and the form field name is Building. Names are case-sensitive.

To add a new target system attribute for trusted reconciliation, follow these steps:

1. In the resource object definition, add a reconciliation field corresponding to the new attribute, as follows:
   a. Open the Resource Objects form. This form is in the Resource Management folder.
   b. Click **Query for Records**.
   c. On the Resource Objects Table tab, double-click the **CONNECTOR_NAME Trusted User** resource object to open it for editing.
   d. On the Object Reconciliation tab, click **Add Field** to open the Add Reconciliation Field dialog box.
   e. Specify a value for the field name that is the name of the new Attribute on your Form.

      For example: Building
   f. From the Field Type list, select a data type for the field.

      For example: String
   g. Save the values that you enter, and then close the dialog box.
   h. If required, repeat Steps d through g to map more fields.
   i. Click **Create Reconciliation Profile**. This copies changes made to the resource object into the MDS.
2. If a corresponding field does not exist in the process form, then add a new column in the process form, as follows:
   a. Open the Form Designer form. This form is in the Development tools folder.
   b. Query for the UD_CONNECTOR_NAMECON form.
   c. Click **Create New Version**. The Create a New Version dialog box is displayed.
   d. In the Label field, enter the name of the version.
   e. Click **Save** and close the dialog box.
   f. From the Current Version box, select the version name that you entered in the Label field in Step 3.
   g. On the Additional Columns tab, click **Add**.
   h. In the Name field, enter the name of the data field and then enter the other details of the field.

      Note: Repeat Steps g and h if you want to add more attributes.
   i. Click **Save** and then click **Make Version Active**.
3. Modify the process definition to include the mapping between the newly added attribute and the corresponding reconciliation field:
   a. Open the Process Definition form. This form is in the Process Management folder of the Design Console.
   b. Click the **Query for Records** icon.
   c. On the Process Definition Table tab, double-click the **CONNECTOR_NAME Trusted User** process definition.
   d. On the Reconciliation Field Mappings tab, click **Add Field Map** to open the Add Reconciliation Field Mapping dialog box.

  e. From the Field Name list, select the name of the resource object that you added in Step 2e.
  f. Double-click Process Data Field and select the corresponding process form field from the Lookup dialog box. Then, click **OK**.
  g. Click **Save** and close the dialog box.
  h. If required, repeat Steps c through g to map more fields.

4. Go to the reconciliation lookup, Lookup.CONNECTOR_NAME.UM.ReconAttrMap.Trusted, and add a new record for the new attribute using the following values:
   - Code Key - Name of the reconciliation field
   - Decode - Name of the CONNECTOR_NAME attribute

10 Understanding ICF Best Practices and FAQs

This chapter enlists the best practices and Frequently Asked Questions (FAQ) on ICF. The list is discussed in the following sections:

- Section 10.1, "Best Practices for ICF"
- Section 10.2, "FAQs on ICF"

10.1 Best Practices for ICF

The following are the best practices that you need to follow while using ICF:

- Use common Scheduled tasks, and ICProvisioningManager.
- Keep IT Resource parameters count to minimum, IT Resource should contain connectivity related parameters only, the rest needs to be in the Main Connector Configuration Lookup.
- Logging in ICF Connectors:

  ICF Integration for Oracle Identity Manager logs all the input/output parameters of all calls to ICF Connector interfaces. You must ensure that the following points are taken care while logging:

  – If required, you can enhance the logging with detailed logging messages.

  – You must not log messages that involves password information or sensitive data.

  – In case you encounter ConnectorException error, then you must wrap the target specific exception, and provide any additional details.

  – Turn on Logging for ICF Common by switching on logging for `oracle.iam.connectors.icfcommon.`
- Connector Load Balancer

  – In order to use SSL-encrypted communication between Oracle Identity Manager and connector servers, you need to copy the SSL keystore on all connector server nodes, and maintain its consistency if SSL key changes.

  – Connector server uses a proprietary (non-HTTP) protocol, and SSL encryption.

  – All connector server nodes under the load balancer should contain the same set of bundles.

10.2 FAQs on ICF

The following are the FAQs on ICF:

- Why lookup reconciliation data contains tilda (~)?

  Tilda (~) notation in lookup reconciliation is to separate Lookups for different IT Resources. In the following example, Key will be a programmatic key, whereas Value will be a user-friendly display name:

  Lookup Key: <IT Resource Key>~<Lookup key>

  Lookup Value: <IT Resource Name>~<Lookup value>

- What is bulk attribute update and how to set it up?

  Bulk attribute update in Oracle Identity Manager means that all the changed attributes will be sent to the connector in one method call, instead of updating each attribute individually (default option).

  In order to enable your connector for bulk attribute update, make sure:

  – all your attributes have their respective process tasks for individual update, typically named as, ATTRIBUTE_NAME updated.

  – you have an extra process task named, UD_FORM_NAME updated. This task will be used for bulk update.

- Search-based versus sync-based reconciliation: when to use what?

  It is based on the capabilities of connector/target resource. Most connectors support search, some of them (LDAP) support sync operation too. Where available, sync-based reconciliation is preferred due to higher efficiency.

  Sync-based reconciliation is more efficient than search-based reconciliation because, it can process both additions/removals in one run. With search-based reconciliation, you need to run search reconciliation first and then run search delete reconciliation, which is double the effort.

- How to configure Connector Pooling?

  See Release 11.1.1.5.0 version of the Connector documentation for information about Connector Pooling and its configuration.

- How to use Groovy to extend connector functionality?

  In order to have an extendable connector, you need to implement `ScriptOnConnector` or `ScriptOnResource` ICF SPIs. Connectors might support various scripting languages, based on target resource capabilities. By default, ICF supports groovy scripts with ScriptOnConnector for all java based connectors. You must always refer the connector documentation to understand the scripting languages for a given connector. See Chapter 9, "Configuring ICF Connectors" for more information about how to customize the connector.

- What are the basic requirements (such as memory, disk space, CPU, and so on) for Connector Server?

  The connector server can run in any Java 6 environment and above. The requirements are same as of those of Java and Oracle Identity Manager.

  See Release 11.1.1.5.0 version of the Connector documentation for the supported versions of JDK and Oracle Identity Manager.

- Does one connector server version support all ICF Connector versions?

Connector Server version equals ICF version. ICF is backward compatible with previously released connector versions.

- How to troubleshoot connector server related issues?

  Set up log level to `FINEST` in logging configuration file of the Connector Server. If the default port 8759 is taken, than set a different port number in the Connector Server configuration.

- When to deploy connector on Connector Server and when to deploy connector locally into Oracle Identity Manager?

  Only .NET connectors require Connectors Server, others can be deployed directly into Oracle Identity Manager.

11 Using Generic Technology Connectors

This chapter describes how to use and maintain Generic Technology Connectors. It contains these sections:

- Overview
- Using the Generic Connection Pool Framework in Custom Connectors
- Best Practices

11.1 Overview

Providers are the starting point for developing generic technology connectors. Oracle Identity Manager provides a standard set of providers that you can use as building blocks of your generic technology connectors. For details about these providers, see Chapter 12, "Predefined Providers for Generic Technology Connectors".

If no suitable provider is available, you can develop a provider to fit your requirements.

Finally, if generic technology connectors do not meet your integration requirements, you can make use of the programmatic options available with adapters. For details, see Chapter 3, "Using the Adapter Factory".

> **See Also:** "Managing Generic Connectors" in *Administering Oracle Identity Manager* for information about creating and managing Generic Technology Connectors

11.2 Using the Generic Connection Pool Framework in Custom Connectors

Custom connectors can choose to use the Generic Connection Pool framework (sometimes referred to as the GCP) for any connection pooling needs.

Internally, the Generic Connection Pool framework uses Oracle Universal Connection Pool (UCP) as the default connection pooling mechanism.

Basic steps to use the Generic Connection Pool in a custom connector include:

1. Provide a concrete implementation for the `ResourceConnection` interface.

   The implementation should also have a default constructor with no parameters.

2. Define the additional fields in the ITResource definition.

3. Invoke the Generic Connection Pool to obtain and release connections from the pool.

Topics in this section include:

- Providing concrete implementation for ResourceConnection interface
- Defining Additional ITResource Parameters
- Getting and Releasing Connections from the Pool
- Using a Third-party Pool
- Example: Implementation of ResourceConnection

11.2.1 Providing concrete implementation for ResourceConnection interface

The connection pool makes use of the concrete implementation of `ResourceConnection` to create and close connections, and to validate connections to the target. Thus, you should ensure that this concrete implementation class is available as a jar file under the `JavaTasks` folder.

Table 11–1 describes key methods of `ResourceConnection`:

Table 11–1 Methods of ResourceConnection

Method	Description
Create Connection	This method is called while initializing the pool (to create initial number of connections) and for pool life-cycle events as needed. A hashmap named `itResourceInfoMap` is available as parameter with ITResource values to this method. The method returns the `ResourceConnection` which is the actual physical connection to the target.
Close Connection	The pool invokes this method when it needs to close a connection in the course of pool life-cycle events.
Heartbeat	This method is used to maintain the TCP heartbeat (or TCP keepalive) of the connection to the target. The method keeps the TCP connection alive, so that the connection does not time out from the target side.
Validate	This method returns `true` or `false` to indicate whether the connection is still valid. The Generic Connection Pool invokes the method if "validate connection on borrow" is set to `true`. It is invoked for connections that have been in the pool for some time. If the method returns `false`, the pool will discard that connection, create a new connection, and return to the requester.

11.2.2 Defining Additional ITResource Parameters

Table 11–2 lists other `ITResource` parameters for which you should provide appropriate values:

Table 11–2 ITResource Parameters

Field	Description	Sample Value and Notes
Abandoned connection timeout	Connection timeout for abandoned connections in seconds. After the timeout elapses, the connection is reclaimed.	900
Connection wait timeout	Wait time in seconds for a connection to establish.	60
Inactive connection timeout	Connection timeout, in seconds, for inactive connections in the pool that are idle. *Note*: These are not borrowed connections.	300
Initial pool size	Initial number of connections in the pool.	3
Max pool size	Maximum number of connections that the pool can create.	30
Min pool size	Minimum number of connections that the pool must maintain.	2
Validate connection on borrow	Indicates if connections should be validated. See Table 11–1 for a detailed explanation.	true or false
Timeout check interval	Frequency, in seconds, at which to check timeout properties.	30
Pool preference	Denotes the preferred pooling mechanism. Default pool implementation is UCP.	"Default" (for UCP). "Native" (for Native implementation)
Connection pooling supported	Denotes whether pooling is supported. If pooling is not supported, returned connections will not be pooled connections. Recommended default is `true`.	true or false.
Target supports only one connection	Denotes whether the target system supports only one connection at a time. When set to true, irrespective of other properties, the following pool parameters are used: ■ Min Pool Size = 0 ■ Initial Pool Size = 0 ■ Max Pool Size = 1 Recommended default is `false`	`true` if target can handle only one connection, `false` otherwise.
ResourceConnection class definition	The concrete implementation of the `ResourceConnection` class	`com.oracle.oim.ad.ADResourceConnectionImpl`
Native connection pool class definition	The wrapper to the native pool mechanism that implements the GenericPool. Set a value only if the pool preference is set to `Native`.	`com.oracle.oim.ad.ADNativePool`
Pool excluded fields	Comma-separated list of fields not needed for creating a connection. When any of the specified fields are updated, the GCP pool is *not* refreshed. Note: Fields in this list are not available as part of the `HashMap` parameter to the `createConnection` method.	`Recon TimeStamp,ADSync Enabled`

Note the following:

- Updating the `ITResource` parameters from the Design Console does not refresh the pool. Update values through the Identity System Administration or through the APIs.

- Avoid updating values when the pool is in use.

11.2.3 Getting and Releasing Connections from the Pool

Consumers of the Generic Connection Pool can invoke the ConnectionService to get pooled connections to the target, and also to return connections back to the pool.

This example code gets a connection from the pool and returns it based on ITResource Name:

```
import com.oracle.oim.gcp.exceptions.ConnectionServiceException;
import com.oracle.oim.gcp.pool.ConnectionService;
import com.oracle.oim.gcp.resourceconnection.ResourceConnection;

public class ConnectionPoolClient {

        public void testConnection(String itResName)
        {
               try{
                         //Request for connection from the connection pool
                         ConnectionService service = new ConnectionService();
                         ResourceConnection myConnection =
                         service.getConnection(itResName);

                         //"myConnection" is the connection
                         //use the connection...

                         //Release connection back to the connection pool
                         //Connections should always be returned this way.

                         service.releaseConnection(myConnection);
                }
                catch(ConnectionServiceException e)
                {
                     //handle
                }
       }
```

You can also request connections to the target using ITResource Key. Here is an example:

```
ConnectionService service = new ConnectionService();
ResourceConnection myConnection = service.getConnection(itResourceKey);
```

11.2.4 Using a Third-party Pool

As mentioned earlier in the section, you can use any third-party pool for your custom connector. However, in addition to the steps described earlier, you must provide a concrete implementation of the GenericPool interface as a wrapper to the third-party pool.

Note: It If the custom connector does not wish to use the UCP pool, it can choose to use GCP with the Native option, though there are no significant advantages to this. With the Native pool preference, the responsibility of maintaining and implementing the pool rests with the custom connector.

Table 11–3 lists the methods invoked for the GenericPool interface:

Table 11–3 Methods of the GenericPool Interface

Method	Purpose
initializePool(PoolConfiguration poolConfig)	To initialize the pool. The PoolConfiguration data object contains all pool-related parameters.
borrowConnectionFromPool()	To request a connection.
returnConnectionToPool(Resource Connection resConn)	To return a connection to the pool.
refreshPool(PoolConfiguration newPoolConfig)	To refresh the pool with updated values.
destroyPool()	To remove the pool (for example when ITResource is deleted).

11.2.5 Example: Implementation of ResourceConnection

This example demonstrates an implementation of the `ResourceConnection` interface. Key methods are highlighted.

Example 11–1 An Example of ResourceConnection Implementation

```
/**
* Sample implementation for Socket Connections:
*/
import java.io.IOException;
import java.net.InetSocketAddress;
import java.net.Socket;
import java.net.SocketException;
import java.net.UnknownHostException;

import com.oracle.oim.gcp.exceptions.ResourceConnectionCloseException;
import com.oracle.oim.gcp.exceptions.ResourceConnectionCreateException;
import com.oracle.oim.gcp.exceptions.ResourceConnectionValidationxception;
import com.oracle.oim.gcp.resourceconnection.ResourceConnection;

public class SocketResourceConnectionImpl extends Socket implements
ResourceConnection {
 public SocketResourceConnectionImpl() {
             super();

       }
       /**
        * Sample: Concrete implementation for closing a socket connection
        */
 public void closeConnection() throws ResourceConnectionCloseException{
               if(!this.isClosed()){
                        try {
                            this.close();
                        } catch (IOException e) {
                            throw new
                            ResourceConnectionCloseException("[Client
                             ResourceConnection implementation]
                            Failed to close socket connection! ");
                        }
               }
        }
        /**
```

```
         * Sample : Concrete implementation for creating a socket connection.
         * The return value is the actual physical socket connection
         *
         */
         public ResourceConnection createConnection(HashMap itResInfoMap)
         throws ResourceConnectionCreateException {
            ResourceConnection r = null;
            SocketResourceConnectionImpl i = new
            SocketResourceConnectionImpl();

              try {
//HashMap has all ITResource related information that is needed
//for connecting to target.
                   String serverAddress= ((String) itResInfoMap.get
                      ("Server Address")).trim();
//utility method getIntValue returns an int for a String

                   int port =
                       getIntValue(((String)itResInfoMap.get("Port")).trim());

                   System.out.println("Connecting to Socket with IP Address "
                                       + serverAddress+" at port "+ port);
                   InetSocketAddress inet = new
                        InetSocketAddress(serverAddress,port);
                        i.connect(inet);
                        if(!i.isConnected()){
                              throw new ResourceConnectionCreateException
                              (" Failed to create socket: connection failure");
}
r = (ResourceConnection)i;
                } catch (UnknownHostException e) {
                        throw new ResourceConnectionCreateException("
                        [Client ResourceConnection implementation]
                        Failed to create socket connection!", e);
                } catch (IOException e) {
                        throw new ResourceConnectionCreateException("
                        [Client ResourceConnection implementation]
                        Failed to create socket connection! ",e);
                }

                return r;
}
 /**
         * Sample : Concrete implementation for heartbeat of a socket connection
         */
        public void heartbeat() throws ResourceConnectionValidationxception {
               try {
                    this.setKeepAlive(true);

               } catch (SocketException e) {
                      throw new
                      ResourceConnectionValidationxception
                      ("[Client ResourceConnection implementation]
                         Failed to set heartbeat ");
               }

        }
 /**
         * Sample: Concrete implementation for validating connection
         */
        public boolean isValid() {
               if(this.isBound()){

                     return true;

               }else{
                     return false;
               }
        }
}
```

11.3 Best Practices

This section contains these topics:

11.3.1 Working with the Provide Basic Information Page

Apply the following guidelines while specifying a name for a generic technology connector:

- **Summary:**

  Ensure that the name contains only ASCII characters. You can include the underscore (_) character, but do not include any other non-ASCII character in the name.

  Description:

  For most of the connector objects that are automatically created at the end of the connector creation process, the name of the generic technology connector is part of the name of the object itself. For example, if the name of the generic technology connector is `WebConn`, the name of its scheduled task is `WebConn_GTC`.

  In the Oracle Identity Manager database, there is no provision for storing objects with names in non-ASCII characters. Therefore, an error message is displayed if you enter non-ASCII characters while specifying the name of a generic technology connector.

- Ensure that the name is not the same as the name of any connector or connector object on the Oracle Identity Manager installation.
- If you plan to create the generic technology connector on a staging server and move it to a production server, ensure that the name of the generic technology connector does not cause naming conflicts with existing connectors or connector objects on the production server.
- Before you import a generic technology connector created on a staging server to a production server, create a backup of the destination Oracle Identity Manager

database to ensure that you are able to revert to a working state in the event that a connector object is overwritten.

- If you select the shared drive transport provider, you must select the CSV format provider.
- If you select the SPML provisioning format provider, you must select the Web Services provisioning transport provider.
- If you select the shared drive reconciliation transport provider, ensure that there is data in the prescribed format on at least the first two lines of the parent and child data files provided by the target system for reconciliation. The prescribed form of data is discussed in Section 12.1, "Shared Drive Reconciliation Transport Provider" .
- If you select the shared drive reconciliation transport provider, ensure that the required permissions are set on the staging and archiving directories before reconciliation begins. The required permissions are discussed in the "Permissions to Be Set on the Staging and Archiving Directories" section .
- Do not try to create more than one generic technology connector at a time, even from different sessions of the Identity System Administration for the same Oracle Identity Manager installation.

11.3.2 Working with the Specify Parameter Values Page

This section describes the following known issues related to the input that you specify on the Step 2: Specify Parameter Values page:

- **Summary:**

  If you use the shared drive reconciliation transport provider, :

  - Do not specify the same path for the staging and archiving directories. Existing files in the archiving directory are deleted if you specify the same path for both directories.
  - Ensure that the names of files in the staging directory are different from the names of files in the archiving directory. If the file names happen to be the same, existing files in the archiving directory are overwritten at the end of a reconciliation run.

  Description:

  When you use the shared drive reconciliation transport provider, after each reconciliation run, data files are moved from the staging directory to the archiving directory. The files moved to the archiving directory are not time-stamped or marked in any way. Therefore, when you use the shared drive transport provider, bear in mind the following guidelines:

  - The archiving directory path and name that you specify must not be the same as the staging directory path and name. If you specify the same path and name, the existing files in the archiving directory are deleted at the end of the reconciliation run.
  - During the current reconciliation run, if data files with the same names as the files used in the last reconciliation run are placed in the staging directory, the existing files in the archiving directory are overwritten by the new files from the staging directory. This can be illustrated by the following example:

    Suppose that at the end of the last reconciliation run, the following files were moved automatically from the staging directory to the archiving directory:

```
usrdataParentData.csv
usrdataRoleData.csv
usrdataGroupMembershipData.txt
```

For the current reconciliation run, you place the following files in the staging directory:

```
usrdataParentData.csv
usrdataRoleData_04Feb07.csv
usrdataGroupMembershipData_04Feb07.txt
```

At the end of the current reconciliation run, these files are moved to the archiving directory. When this happens, the old `usrdataParentData.csv` file is overwritten by the new one.

Therefore, if you want to ensure that files in the archiving directory are not overwritten at the end of a reconciliation run, you must ensure that the names of files in the staging directory are not the same as the names of files in the archiving directory.

- **Summary:**

  Metadata detection does not take place a second time if you go back to the Step 2: Specify Parameter Values page. Therefore, if required, you must manually make changes in the fields and field mappings displayed on the Step 3: Modify Connector Configuration page.

  Description:

  Suppose you want to change a value that you provide on the Step 2: Specify Parameter Values page. You can return to this page from the Step 4: Verify Connector Form Names or Step 5: Verify Connector Information page. However, metadata detection would not take place a second time when you click the Continue button after changing the provider parameter value. This functionality is aimed at preserving changes that you may have manually made on the Step 3: Modify Connector Configuration page.

  As an extension of this functionality, metadata detection does not take place even when you modify an existing generic technology connector.

11.3.3 Working with the Modify Connector Configuration Page

This section discusses best practices related to the following areas:

- Names of Fields
- Password Fields
- Password-Like Fields
- Mappings
- Oracle Identity Manager Data Sets

11.3.3.1 Names of Fields

Note that the following validations are applied when you specify a field name while adding or editing fields:

- Two fields that belong to the same data set (parent or child) cannot have the same name.
- Two child data sets of the same parent data set cannot have the same name.

- The name of a field in a parent data set cannot be the same as the name of one of its child data sets.
- Two different child data sets can have fields that have the same name, regardless of whether or not the child data sets belong to the same parent data set. For example, the `GroupMembership` data set and `Role` data set can each have a field with the name `UsrID`.
- Two different parent data sets can have fields that have the same name. Similarly, these data sets can also have child data sets that have the same name.
- The name of a child data set can be the same as that of one of its fields.

11.3.3.2 Password Fields

To ensure the security of passwords, password information must not be reconciled through a generic technology connector. In other words, you must ensure that the Source and reconciliation staging data sets do not contain the Password field. In addition, you must not map any field of the reconciliation staging data sets to the Password field of the OIM - User data set.

11.3.3.3 Password-Like Fields

A password-like field is a field to which you set the Encrypted and Password Field attributes (by selecting the Encrypted and Password Field check boxes). You can create a password-like field by setting these two attributes to a field that you add to the OIM - Account data set.

To secure the contents of password-like fields, bear in mind the following guidelines while adding or editing these fields:

- You can use the Password Field and Encrypted check boxes to secure the display and storage of password information in Oracle Identity Manager. However, when you map password-like fields to fields of the provisioning staging data set, you must take all necessary precautions to secure the data propagated in these fields. For example, you must ensure that this data is not stored in a plain-text file on the target system at the end of a provisioning operation.

  Oracle recommends creating only one-to-one mappings between the password field of the OIM - Account data set and the provisioning staging data set. In other words, this password field must not be used as an input field for a transformation mapping with a provisioning staging field. The same precaution must be taken for the Password field of the OIM - User data set.
- As mentioned earlier, the Password field is one of the predefined fields of the OIM - User data set. The Password Field and Encrypted attributes are set for this field. By using the Design Console, you can set the Password Field and Encrypted attributes for a UDF that you create. This would give the newly created UDF the same properties as the existing Password field. However, the generic technology connector framework treats this field the same as any other text field (with the String data type) and the contents are not encrypted in the Identity Self Service, Identity System Administration, or database.

11.3.3.4 Mappings

Apply the following best practices while working with fields of the Oracle Identity Manager data sets:

- **Summary:**

If you select the translation transformation provider to create a mapping, specify the name of a lookup definition in the Lookup Code Name region. If you specify a data set name and field in the Lookup Code Name region, translation would fail during actual reconciliation or provisioning operations.

Description:

If you select the translation transformation provider while creating a mapping, the Step 2: Mapping page displays options for selecting a field from a data set and specifying a literal. Because you are using the translation transformation provider, you must select the Literal option and enter the name of the lookup definition that contains the Code Key and Decode values for the translation. You must not select a data set name and field in the Lookup Code Name region. Although there is no validation to stop you from selecting a data set name and field, the translation operation would fail during actual reconciliation or provisioning operations.

- Create a mapping between the ID field of the OIM - Account data set and the field that uniquely identifies records of the reconciliation staging data set.
- Along with the ID field, other fields of the OIM - Account data set can be (matching-only) mapped to corresponding fields of the reconciliation staging data set to create a composite key field for reconciliation matching.
- Create mappings between all fields in provisioning staging data sets and corresponding fields in Oracle Identity Manager data sets.
- To create a reconciliation rule, you create matching-only mappings between fields of the reconciliation staging data set and the OIM - User data set. If there are child data sets, ensure that the names of fields of the reconciliation staging data set that are input fields for the matching-only mappings are not used in any of the reconciliation staging child data sets. If you do not follow this guideline, reconciliation would fail.

  This has also been mentioned in the section "Step 3: Modify Connector Configuration Page".
- A literal field can be used as one of the input fields of a transformation mapping. If you select the Literal option, you must enter a value in the field. You must not leave the field blank after selecting it.

11.3.3.5 Oracle Identity Manager Data Sets

Apply the following best practices while working with fields of the Oracle Identity Manager data sets:

- For trusted source reconciliation, the following fields of the OIM – User data set must always hold values:
  - User ID
  - First Name
  - Last Name
  - Organization Name
  - Xellerate Type
  - Role

  In addition, you can select other OIM – User fields that must be populated when a user account is created through reconciliation. For each of these fields, you must create mappings with the corresponding fields of the reconciliation staging data sets. During a reconciliation run, you must ensure that the fields of the target system that serve as the source for these fields always hold values.

For provisioning, you can select fields of the OIM – User and OIM – Account data sets whose values must be propagated to the target system. After you identify these fields, create mappings between them and their corresponding fields in the provisioning staging data sets. During a provisioning operation, you must enter values for each of these fields.

- If required, add user-defined fields (UDFs) to the list of predefined OIM - User data set fields.
- Do not modify or delete attributes of OIM - Account data set fields in an existing generic technology connector.

11.3.4 Working with Shared Drive Reconciliation Transport Provider

Summary

If parent records and child data records are created and linked in both the target system and Oracle Identity Manager, you must ensure that the staging directory contains both parent data and child data files at the start of each reconciliation run.

Description

Suppose there are parent data records with associated child data records in the target system. To reconcile these records into Oracle Identity Manager, you place the parent and child data files containing these records in the staging directory. During the reconciliation run, the child data records are linked to their corresponding parent data records. Before the start of any subsequent reconciliation run, if you remove the child data files from the staging directory, reconciliation events are not created for this form of child data record deletion. If you want to remove child data records for specific parent data records, you must remove the child data records from the child data file. You must ensure that the child data file is placed in the staging directory for each reconciliation run, even if there are no child data records (from the third line onward) in the files.

11.3.5 Working with Custom Providers

Apply the following guideline while working with custom providers:

When you develop code for a custom provisioning transport provider, ensure that the provider returns the unique field value at the end of a Create User operation. This functionality is implemented by the `sendData` method of the provisioning transport provider. See "Role of Providers During Provisioning" for more information.

11.3.6 Working with Connector Objects

Apply the following guidelines while working with connector objects created automatically during generic technology connector creation:

- **Summary:**

  Do not attempt to use for provisioning the resource object created automatically for a reconciliation-only generic technology connector.

  Description:

  Suppose you select only the Reconciliation option while creating a generic technology connector. At the end of the creation process, a resource object is one of the objects created automatically for this generic technology connector. However, you cannot provision this resource object to any user because a generic adapter is not created for a reconciliation-only generic technology connector.

- **Summary:**

  Do not attempt to provision generic technology connector resource objects to organizations defined in Oracle Identity Manager.

  Description:

  A resource object is one of the connector objects that get created automatically during generic technology connector creation. This resource object can be provisioned only to Oracle Identity Manager Users. You must not attempt to provision it to organizations defined in Oracle Identity Manager.

  This has also been mentioned in the Connector Objects section .

- You can use the Design Console to customize connector objects that are automatically created during generic technology connector creation. After you customize connector objects, if you perform a Manage Generic Technology Connector operation, all the customization done on the connector objects would be overwritten. Therefore, Oracle recommends that you to apply one of the following guidelines:

  – Do not use the Design Console to modify generic technology connector objects.

    The exception to this guideline is the IT resource. You can modify the parameters of the IT resource by using the Design Console. However, if you have enabled the cache for the `GenericConnector` and `GenericConnectorProviders` categories, you must purge the cache either before or after you modify IT resource parameters.

  – If you use the Design Console to modify generic technology connector objects, do not use the Manage Generic Technology Connector feature to modify the generic technology connector.

  This has also been mentioned in Section 11.3.6, "Working with Connector Objects".

- Prepopulate adapters are not part of the set of connector objects that are created automatically when you create a generic technology connector. However, while creating a generic technology connector, you can map provisioning input to literals and user data fields. Although this feature cannot be used to prepopulate the User Defined Form, it can be used to prepopulate the provisioning data packet.

11.3.7 Modifying Generic Technology Connectors

Apply the following best practice while modifying generic technology connectors:

Do not try to modify more than one generic technology connector at a time, even from different sessions of the Identity System Administration for the same Oracle Identity Manager installation.

12

Predefined Providers for Generic Technology Connectors

The following predefined providers are shipped Oracle Identity Manager:

> **Note:** You must determine the values of parameters for providers that you decide to use. You would need to use these values while creating the generic technology connector by using Oracle Identity System Administration.

- Shared Drive Reconciliation Transport Provider
- CSV Reconciliation Format Provider
- SPML Provisioning Format Provider
- Web Services Provisioning Transport Provider
- Transformation Providers
- Validation Providers

12.1 Shared Drive Reconciliation Transport Provider

The shared drive reconciliation transport provider reads data from flat files stored in staging directories and moves the files to an archiving directory. The staging and archiving directories must be shared for access from the Oracle Identity Manager server.

The following are parameters of this provider:

- **Staging Directory (Parent identity data)**

  Use this parameter to specify the path of the directory in which files containing parent data are stored. It is mandatory to specify a value for this parameter. This is a run-time parameter.

  In this guide, **parent data** means the user account information that is stored in the target system.

  Sample value for this parameter:

  ```
  T:/TargetSystemDirectory/ParentData
  ```

Note: If the staging directory is not on the server on which Oracle Identity Manager is installed, it must be shared and mapped as a network drive on the Oracle Identity Manager server.

Data stored in the parent data files must conform to the following conventions:

– First line of the file

  The first line of the parent data file must be the file header that describes the contents of the file.

  The file header can be preceded by any number of lines that begin with the hash-mark or pound-sign (#). These are ignored while the file is read. However, you must ensure that there are no spaces at the start of the header. If you are using a language other than English, you must not enter non-ASCII characters on this line.

Note: There are no checks to stop you from entering non-ASCII characters on the first line. In addition, the generic technology connector framework can parse such characters. However, the use of non-ASCII characters would result in problems at the time when the connector objects are automatically created for the generic technology connector that you create.

– Second line of the file

  The second line of the parent data file must contain the field names (metadata) for the data in the file.

Note: In the generic technology connector context, the term **metadata** refers to the set of identity fields that constitute the user account information.

  If you are using a language other than English, you must not enter non-ASCII characters on this line. See the Note in the preceding point for more information about this limitation.

– Third line of the file onward

  From the third line onward, the parent data file can contain data in the language that you have selected for Oracle Identity Manager. This language can have an ASCII or non-ASCII character set.

  Even if there is no data from the third line onward, reconciliation will take place and the files are archived.

The following are contents of a sample parent data file:

```
##Active Directory user
Name TD,Address TD,User ID TD
John Doe,Park Street,jodoe
Jane Doe,Mark Street,jadoe
```

See Also: "Permissions to Be Set on the Staging and Archiving Directories"

- **Staging Directory (Multivalued identity data)**

  Use this parameter to specify the path of the directory in which files containing multivalued (or child) account or identity data (for example, role membership data) are stored. It is *not* mandatory to specify a value for this parameter. This is a run-time parameter.

  > **Note:** In this guide, the terms multivalued account or identity data and child data have been used interchangeably.

  Sample value for this parameter:

  ```
  T:/TargetSystemDirectory/ChildData
  ```

  > **Note:**
  >
  > - The staging directory for parent data files cannot be the same as the staging directory for multivalued user data files. In addition, if the staging directory is not on the same server on which Oracle Identity Manager is installed, it must be shared and mapped as a network drive on the Oracle Identity Manager server.
  > - If you select the Trusted Source Reconciliation option on the Step 1: Provide Basic Information page, you must not specify a value for the Staging Directory (Multivalued Identity Data) parameter. This is because the reconciliation of multivalued (child) data is not supported in trusted source reconciliation.

  For each type of multivalued account or identity data, there must be a different file in the shared directory. For example, if the multivalued user data for a particular target system is group membership data and role data, there must be one file for group membership data and a different file for role data.

  Data stored in the child data files must conform to the conventions (first line, second line, and remaining lines) that are specified for the parent data files.

  In addition, the same unique field must be present in the parent data file and each child data file. This field is used to uniquely link each record in the child data files with a single record in the parent data file. This structure is similar to the concept of integrity constraints (primary key-foreign key) in RDBMSs.

  > **Note:** The unique field must be the first field in the child data files.

  The following are contents of a sample child data file holding role information that is linked to the sample parent data file listed earlier:

  ```
  ###Role
  User ID TD,Role Name TD,Role Type TD
  jodoe,admin1,admin
  jadoe,admin2,admin
  ```

  The following are contents of a sample child data file holding group membership information that is linked to the sample parent data file listed earlier:

  ```
  ###Group Membership
  User ID TD,Group Name TD,Group Type TD
  ```

```
jodoe,OracleDev1,OracleDev
jadoe,OracleDev2,OracleDev
jadoe,OracleDev3,OracleDev
jadoe,OracleDev4,OracleDev
jadoe,OracleDev5,ConnectorDev
```

Note that the name of the unique field, `User ID TD`, is the same in the child data files and the parent data file.

On the Step 3: Modify Connector Configuration page as described in "Step 3: Modify Connector Configuration Page" in *Administering Oracle Identity Manager*, the name of a child data set is the same as the header that you provide in the child data file. For these sample child data files, the child data sets would be labeled `Role` and `Group Membership`. In addition, on the Step 4: Verify Connector Form Names page, the default names displayed for forms corresponding to the child data sets would be `Role` and `Group Membership`. As mentioned in "Step 4: Verify Connector Form Names Page" in *Administering Oracle Identity Manager*, you can either accept the default form names or change them.

> **See Also:** "Permissions to Be Set on the Staging and Archiving Directories"

- **Archiving Directory**

  Use this parameter to specify the path of the directory in which parent and child data files that have already been reconciled are to be stored. This is a run-time parameter.

  It is mandatory to specify a value for this parameter.

  At the end of the reconciliation run, the data files are copied into the archiving directory and deleted from the staging directory.

  The files moved to the archiving directory are not time stamped or marked in any way. Therefore, while specifying the path of the archiving directory, bear in mind the following guidelines:

  – The archiving directory path that you specify must not be the same as the staging directory path. If you specify the same path, the existing files in the archiving directory are deleted at the end of the reconciliation run.

  – If data files with the same names as the files used in the last reconciliation run are placed in the staging directory, the existing files in the archiving directory are overwritten by the new files from the staging directory at the end of the current reconciliation run.

  These points are also mentioned in "Step 2: Specify Parameter Values Page" in *Administering Oracle Identity Manager*.

  > **See Also:** "Permissions to Be Set on the Staging and Archiving Directories"

- **File Prefix**

  Use this parameter to specify the prefix used to filter the names of files in the staging directories for both parent and child data files. During reconciliation, all files (in the staging directories) with names that start with the specified prefix are processed, regardless of the file extension. This is a run-time parameter.

  For example:

If you specify `usrdata` as the value of the File Prefix parameter, data is parsed from the following files placed in the staging directory for multivalued (child) user data files:

```
usrdataRoleData.csv
usrdataGroupMembershipData.txt
```

Data is not extracted from the following files in the same directory, because the file names do not begin with `usrdata`:

```
RoleData.csv
GroupMembershipData.txt
```

- **Specified Delimiter**

  Use this parameter to specify the character that is used as the delimiter character in the parent and child data files. You can specify only a single character as the value of this parameter. This is a run-time parameter. This parameter overrides the Tab Delimiter parameter.

  Note: You cannot use the space character ( ) as a delimiter.

  In addition, you must ensure that the character you specify is used only as the delimiter in the data files. If this character is also used inside the data itself, the data row (or record) is not parsed correctly. For example, you must not use the comma (,) as the delimiter if any data value contains a comma.

- **Tab Delimiter**

  Use this parameter to specify whether or not the file is delimited by tabs. This is a run-time parameter. This parameter is ignored if you specify a value for the Specified Delimiter parameter.

- **Fixed Column Width**

  If the input file contains fixed-width data, use this parameter to specify the width in characters of the data columns. This is a run-time parameter.

  Note: In this context, the term "fixed-width" refers to the number of characters in the data field, not the byte length of the field. This means that, for example, four characters of single-byte data and four characters of multibyte data are the same in terms of width.

  This parameter is ignored if you specify a value for the Specified Delimiter or Tab Delimiter parameter.

- **Unique Attribute (Parent Data)**

  For multivalued user data, use this parameter to specify the field that is common to both the parent data and child data files. In the examples described earlier, the requirement for a unique attribute is fulfilled by the `User ID TD` field, which is present in both the parent and child data files. This is a run-time parameter.

Note: If you select the Trusted Source Reconciliation option on the Step 1: Provide Basic Information page, you must not specify a value for the Unique Attribute (Parent Data) parameter. This is because the reconciliation of multivalued (child) data is not supported in trusted source reconciliation.

- **File Encoding**

  Use this parameter to specify the character set encoding used in the parent and data files. This is a design parameter.

  Specify `Cp1251` for data files stored on a computer running an operating system with the English-language setting. This is the canonical name for the `java.io` API that is supported by the generic technology connector framework. For any other language that you select from the list given in the "Multilanguage Support" section, you must specify the canonical name for the corresponding `java.io` API.

Permissions to Be Set on the Staging and Archiving Directories

You must ensure that the required permissions are set on the staging and archiving directories. The following table describes the effect of the various permissions on the shared directories that are used to hold staging and archiving data files.

Storage Entity	Access Permission	Reason for Access Permission Requirement
Staging directory for parent data files	Read	This permission is required for reconciliation to take place. An error message is logged if this permission is not applied.
Staging directory for parent data files	Write	This permission is required for the deletion of data files from the parent staging directory at the end of the archive process.
Staging directory for parent data files	Execute	Not applicable
Staging directory for child data files	Read	This permission is required for the reconciliation of child data. An error message is logged if this permission is not applied.
Staging directory for child data files	Write	This permission is required for the deletion of data files from the child staging directory at the end of the archive process.
Staging directory for child data files	Execute	Not applicable
Archiving directory	Write	This permission is required for the copying of parent and child data files to the archiving directory during the archive process. Even if this permission is not applied: ■ Parent and child data reconciliation takes place. ■ Files are deleted from the parent and child staging directories if the required permissions have been set on those directories.
Archiving directory	Execute	Not applicable
Parent or child data file in staging directory	Read	This permission is required for the reconciliation of the data in the file. An error message is logged if this permission is not applied.
Parent or child data file in staging directory	Write	This permission is required for the deletion of the data file at the end of the archive process. An error message is logged if this permission is not applied. However, data in this file is reconciled.

Storage Entity	Access Permission	Reason for Access Permission Requirement
Parent or child data file in staging directory	Execute	Not applicable

Note: Data files in the staging directory cannot be deleted if they are open in any editor or are open for writing by any other program.

12.2 CSV Reconciliation Format Provider

The CSV reconciliation format provider converts reconciliation data that is in character-delimited, tab-delimited, or fixed-length format into a format that is supported by Oracle Identity Manager.

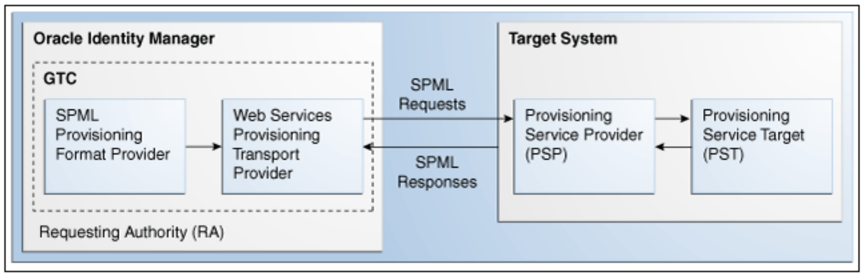
Although the CSV reconciliation format provider is packaged as a standalone provider, all of its parameters are bundled with the shared drive transport provider. If you select the shared drive transport provider on the Step 1: Provide Basic Information page, you must select the CSV format provider. When you select this provider, its parameters are displayed along with the shared drive transport provider parameters.

12.3 SPML Provisioning Format Provider

The SPML provisioning format provider converts the provisioning data generated during a provisioning operation on Oracle Identity Manager into an SPML request that can be processed by an SPML-compatible target system.

Figure 12–1 shows the setup of the system in which the SPML provisioning format provider acts as the requesting authority (RA), and the target system provides the provisioning service provider (PSP) and the provisioning service target (PST).

Figure 12–1 Communication Between the SPML Provisioning Format Provider and the Target System


Oracle Identity Manager
GTC
SPML Provisioning Format Provider
Web Services Provisioning Transport Provider
Requesting Authority (RA)
SPML Requests
SPML Responses
Target System
Provisioning Service Provider (PSP)
Provisioning Service Target (PST)

During actual provisioning, a Velocity template engine is used to create the SOAP-SPML requests. For the following processes, the provider generates SOAP requests based on the SPML 2.0 DSML profile:

- Add request
- Modify request for the following Oracle Identity Manager process tasks:
  - Field updated
  - Add child data
  - Modify child data

  - Delete child data
- Suspend request (for Disable Oracle Identity Manager process tasks)
- Resume request (for Enable Oracle Identity Manager process tasks)
- Delete request

The Create Organization, Update Organization, and Delete Organization are not supported. This is because the resource object created for a generic technology connector does not support provisioning operations for organizations. The Create Group, Update Group, and Delete Group operations are not supported. This is because Oracle Identity Manager does not support operations to provision groups.

When you select this provider, the following identity fields are displayed by default on the Step 3: Modify Connector Configuration page as described in "Step 3: Modify Connector Configuration Page" in *Administering Oracle Identity Manager*, along with the `ID` field:

- `objectClass`
- `containerID`

For each provisioning task (for example, Create User and Modify User), the provider generates a request in a predefined format.

The following sections discuss the parameters of this provider:

- Run-Time Parameters
- Design Parameters

Depending on the application server that you use, some of the run-time and design parameters are mandatory and some have fixed values. The following sections discuss these parameters:

- Nonmandatory Parameters
- Parameters with Predetermined Values

12.3.1 Run-Time Parameters

The following are run-time parameters of the SPML provisioning format provider:

- **Target ID**

  This value uniquely identifies the target system for provisioning operations.

- **User Name (authentication)**

  This is the user name of the account required to connect to the target system (PST) through the Web service interface (PSP).

- **User Password (authentication)**

  This is the password of the user account required to connect to the target system (PST) through the Web service interface (PSP).

12.3.2 Design Parameters

The following are design parameters of the SPML provisioning format provider:

> **See Also:** For more information about the SOAP elements and attributes mentioned in this section, visit the following Web site
>
> `http://www.w3.org/TR/wsdl20/`

- **Web Service SOAP Action**

  In the WSDL file, this is the value of the `soapAction` attribute of the `operation` element.

- **WSSE Configured for SPML Web Service?**

  Select this check box if the Web service is configured to authenticate incoming requests by using WS-Security credentials.

- **Custom Authentication Credentials Namespace**

  > **Note:** You need not specify a value for this parameter if you select the SPML Web Service WSSE Configured? check box.

  This is the name of the credentials namespace that you have defined for the Web service. In most cases, this namespace is the same as the target namespace.

- **Custom Authentication Header Element**

  > **Note:** You need not specify a value for this parameter if you select the SPML Web Service WSSE Configured? check box.

  This is the name of the element that will contain the credentials of the user account used to connect to the target system. In other words, this is the parent element in the custom authentication section of the SOAP message header.

- **Custom Element to Store User Name**

  > **Note:** You need not specify a value for this parameter if you select the SPML Web Service WSSE Configured? check box.

  This is the name of the element in the custom authentication section that will contain the user name you specify as the value of the User Name (authentication) parameter.

- **Custom Element to Store Password**

  > **Note:** You need not specify a value for this parameter if you select the SPML Web Service WSSE Configured? check box.

  This is the name of the element in the custom authentication section that will contain the user name you specify as the value of the User Password (authentication) parameter.

- **SPML Web Service Binding Style (DOCUMENT or RPC)**

  In the WSDL file, this is the value of the `style` attribute of the `binding` element. You must enter either `DOCUMENT` or `RPC`.

  > **Note:** You must enter the value `DOCUMENT` or `RPC`. Do not use lowercase letters in the value that you specify.

- **SPML Web Service Complex Data Type**

In the WSDL file, this is the value of the `name` attribute of the `complexType` element. This parameter is applicable only if the binding style is `DOCUMENT`. You must specify a value for this parameter if the target Web service is running on Oracle WebLogic Server.

- **SPML Web Service Operation Name**

  In the WSDL file, this is the value of the `name` attribute of the `operation` element. This parameter is applicable only if the binding style is `RPC`.

- **SPML Web Service Target Namespace**

  In the WSDL file, this is the value of the `targetNamespace` attribute of the `definition` element.

- **SPML Web Service Soap Message Body Prefix**

  This is the name of the custom prefix element that contains the SOAP message body. If the target Web service is running on Oracle WebLogic Server, then you need not specify a value for this parameter. However, if you are using a different application server, you must enter the name of the custom prefix element. The following is the prefix element if the Web service is running on Oracle WebLogic Server:

  ```
  <SPMLv2Document xmlns="http://xmlns.oracle.com/OIM/provisioning">
  ```

- **ID Attribute for Child Dataset Holding Group Membership Information**

  This is the name of the unique identifier field for a provisioning staging child data set that holds group membership information. For provisioning operations on the child data set that contains this field, the SOAP packet will contain SPML code for group operations. The following is an SPML code block for this type of group operation:

  ```
  <modification modificationMode="add">
    <capabilityData capabilityURI="urn:oasis:names:tc:SPML:2:0:reference"
  mustUnderstand="true">
      <reference typeOfReference="memberOf"
  xmlns="urn:oasis:names:tc:SPML:2:0:reference">
      <toPsoID ID="Groups:1" targeted="120"/>
  </reference>
    </capabilityData>
  </modification>
  ```

  For provisioning operations on the child data sets that do not contain this field, the SOAP packet will contain ordinary SPML code. The following is an SPML code block for this type of group operation:

  ```
  <modification>
    <dsml:modification name="Group Membership" operation="add">
      <dsml:value>AdminOra, System Admins, USA</dsml:value>
    </dsml:modification>
  </modification>
  ```

12.3.3 Nonmandatory Parameters

For Oracle WebLogic Server, you need not specify values for the following parameters:

- SPML Web Service Complex Data Type
- SPML Web Service Soap Message Body Prefix

- ID Attribute for Child Dataset Holding Group Membership Information

12.3.4 Parameters with Predetermined Values

For Oracle WebLogic Server, you can specify predetermined values for the following parameters:

- Web Service URL: `http://`*`IP_address`*`:`*`port_number`*`/spmlws/OIMProvisioning`
- SPML Web Service Binding style (DOCUMENT or RPC): `RPC`
- SPML Web Service Operation Name: `processRequest`

12.4 Web Services Provisioning Transport Provider

The Web Services provisioning transport provider acts as a Web service client and carries provisioning request data from Oracle Identity Manager to the target system Web service.

The following types of target system Web services are supported:

- RPC-literal
- RPC-encoded
- DOCUMENT-literal

The following is the parameter of the Web Services provisioning transport provider:

Web Service URL

Use this parameter to specify the URL of the Web service that you want to use for sending a provisioning request to the target system. This is a run-time parameter. In the WSDL file, the Web service URL is the value of the `location` attribute of the `wsdlsoap:address` element.

If you include the Web Services provisioning transport provider in the generic technology connector that you create, you may want to configure Secure Sockets Layer (SSL) communication between the target system and Oracle Identity Manager. The following section provides information about this procedure.

12.4.1 Configuring SSL Communication Between Oracle Identity Manager and the Target System Web Service

This section describes the procedure to configure the application server on which Oracle Identity Manager is installed for SSL communication.

You can perform this procedure only if all the following conditions are true:

- You want to include the Web Services provisioning transport provider in the generic technology connector that you plan to create.
- The target Web service is running on an SSL-enabled application server.

To configure SSL communication between Oracle Identity Manager and the target system Web service:

Note: You can perform this procedure prior to creating the generic technology connector.

1. Export the target application server certificate as follows:

- For a target system Web service deployed on JBoss Application Server, Oracle WebLogic Server, or Oracle WebLogic Server, run the following command:

  ```
  JAVA_HOME/jre/bin/keytool -export -alias default -file
  exported-certificate-file -keystore app-server-specific-keystore
  -storetype jks -storepass keystore-password -provider
  sun.security.provider.Sun
  ```

  In this command:

  * Replace *JAVA_HOME* with the full path to the SUN JDK directory.
  * Replace *exported-certificate-file* with the name of the file in which you want the exported certificate to be stored.
  * Replace *app-server-specific-keystore* with the path to the keystore on the application server.
  * Replace *keystore-password* with the password for the keystore.

- For a target system Web service deployed on Oracle WebLogic Server on AIX, run the following command:

  ```
  JAVA_HOME/jre/bin/keytool -export -alias default -file
  exported-certificate-file -keystore app-server-specific-keystore -storetype
  jks -storepass keystore-password -provider com.ibm.crypto.provider.IBMJCE
  ```

  In this command:

  * Replace *JAVA_HOME* with the full path to the IBM JDK directory.
  * Replace *exported-certificate-file* with the name of the file in which you want the exported certificate to be stored.
  * Replace *app-server-specific-keystore* with path to the keystore on the application server.
  * Replace *keystore-password* with the password for the keystore.

  When the command is run, the exported certificate file is stored in the file that you specify as the value of *exported-certificate-file*.

2. Import the certificate file exported in the preceding step into the Oracle Identity Manager truststore as follows:

   a. Copy the certificate file exported in the preceding step into a temporary directory on the Oracle Identity Manager server.

   b. Run the following command:

   ```
   JAVA_HOME/jre/bin/keytool -import -trustcacerts -alias servercert -noprompt
   -keystore OIM_HOME\config\.xlkeystore -file certificate_file
   ```

   In this command:

   – Replace *JAVA_HOME* with full path to the JDK directory. For Oracle Identity Manager deployed on Oracle WebLogic Server, the path must be that of the SUN JDK directory.

   – Replace *OIM_HOME* with the full path of the Oracle Identity Manager home directory

   – Replace *certificate_file* with the path of the temporary directory into which you copy the certificate file.

Note: If the application server is enabled for one-way SSL communication, you need not perform the rest of this procedure.

3. Import the Oracle Identity Manager certificate into the target system application server truststore as follows:

   Note: Perform the following steps only if the application server is enabled for two-way SSL communication.

   a. Export the Oracle Identity Manager certificate file.

      For Oracle Identity Manager deployed on Oracle WebLogic Server, run the following command:

      ```
      JAVA_HOME/jre/bin/keytool -export -alias xell -file
      OIM_HOME\config\xell.cert -keystore OIM_HOME\config\.xlkeystore -storetype
      jks -provider sun.security.provider.Sun
      ```

      In this command:

      - Replace *JAVA_HOME* with the full path to the SUN JDK directory.

      - Replace *OIM_HOME* with the full path of the Oracle Identity Manager home directory.

   b. Import the certificate file that you export in Step 3a into the truststore of the application server as follows:

      Copy the exported Oracle Identity Manager certificate file to a temporary directory on the target application server.

      Next, run the following command on the target application server, which is Oracle WebLogic Server:

      ```
      JAVA_HOME/jre/bin/keytool -import -alias alias -trustcacerts  -file
      OIM-certificate-file -keystore app-server-specific-truststore  -storetype
      jks -storepass truststore-password -provider sun.security.provider.Sun
      ```

      In this command:

      * Replace *JAVA_HOME* with the full path to the SUN JDK directory.

      * Replace *alias* with an alias for the certificate in the truststore of the target application server.

      * Replace *OIM-certificate-file* with the name of the exported Oracle Identity Manager certificate file.

      * Replace *app-server-specific-truststore* with path to the truststore on the target application server.

      * Replace *truststore-password* with the password for the truststore on the target application server.

      See Also: SSL configuration documentation for the target application server

12.5 Transformation Providers

Note: Use the information provided in this section while performing the instructions given in "Step 3: Modify Connector Configuration Page" in *Administering Oracle Identity Manager*.

A transformation provider is used to transform user data while it is in transit between the source and destination data sets listed in the following table.

Source Data Set	Destination Data Set	Purpose of the Transformation
Source	Reconciliation Staging	Data is transformed before it is used to create reconciliation events.
Oracle Identity Manager	Provisioning Staging	Data is transformed before it is used to create the provisioning request to be sent to the target system.

The following predefined transformation providers are available in Oracle Identity Manager:

- Concatenation Transformation Provider
- Translation Transformation Provider

12.5.1 Concatenation Transformation Provider

You use the concatenation transformation provider to concatenate the values of two fields of data sets to create the input for a single field of another data set.

The following example explains the output format of this provider:

Suppose the input values are the following fields of the source data set:

- First Name: `John`
- Last Name: `Doe`

When the concatenation transformation provider is applied to these two fields, the output value is as follows:

```
John Doe
```

Note: As shown in the preceding example, the concatenation transformation provider adds a space between the values of the two input fields.

The following procedure describes how to add a concatenation transformation provider while creating a generic technology connector:

Note: This procedure explains in detail the instruction given in Step 5 of "Adding or Editing Fields in Data Sets" in *Administering Oracle Identity Manager*. It is assumed that you have already selected the **Concatenation** option from the **Mapping Action** list on the Step 1: Field Information page and that you have performed Steps 2 and 3 given in that section.

On the Step 2: Mapping page in the pop-up window, perform the following steps:

1. From the **Dataset** list in the Input 1 region, select the data set containing the first field that you want to concatenate. From the **Field Name** list, select the first field. Alternatively, you can use the **Literal** option to specify a literal (or fixed) value as the first concatenation input.

   For the example described earlier, from the **Dataset** list in the Input 1 region, select the data set containing the **First Name** field. Then, from the **Field Name** list, select **First Name**.

2. From the **Dataset** list in the Input 2 region, select the data set containing the second field that you want to concatenate. Then, from the **Field Name** list, select the second field. Alternatively, you can use the **Literal** option to specify a literal (or fixed) value as the second concatenation input.

   For the example described earlier, from the **Dataset** list in the Input 2 region, select the data set containing the **Last Name** field. Then, from the **Field Name** list, select **Last Name**.

12.5.2 Translation Transformation Provider

A translation operation involves accepting a certain (literal) value as input and converting it into another value.

The following example illustrates a translation operation:

Suppose the Source data set contains the Country field and data values stored in this field can take one of the following values:

- Austria
- France
- Germany
- India
- Japan

When these values are propagated to the reconciliation staging data set, you want to convert these values to the following:

- AT
- FR
- DE
- IN
- JP

To automate this translation, you can use the translation transformation provider.

To use the translation transformation provider:

1. Use the Design Console to create a lookup definition that stores the input and decoded values.

   Note: While creating a lookup definition in the Lookup Definition form, you must select the Lookup Type option, and not the Field Type option.

For the Country field example described earlier, the Code Key and Decode values are as shown in the following table.

Code Key	Decode
Austria	AT
France	FR
Germany	DE
India	IN
Japan	JP

2. Define a transformation (translation) mapping between the input field and output field for the translation. As mentioned earlier, a transformation can be set up between the following pairs of data sets:

   - Source and Reconciliation Staging
   - Oracle Identity Manager and Provisioning Staging

   Note: This procedure explains in detail the instruction given in Step 5 of "Adding or Editing Fields in Data Sets" in *Administering Oracle Identity Manager*. It is assumed that you have already selected the **Concatenation** option from the **Mapping Action** list on the Step 1: Field Information page and that you have performed Steps 2 and 3 given in that section.

   a. On the Step 3: Mapping page, from the **Dataset** list in the Input region, select the data set containing the field that will provide the input value for the translation operation. Then, from the **Field Name** list, select the field itself.

      For the Country field example described earlier, select the data set containing the Country field and select the Country field.

   b. In the Lookup Code Name region, select **Literal** and enter the name of the lookup definition that you create in the preceding step.

      Note: You must not specify a data set name and field in the Lookup Code Name region. Although there is no validation to stop you from selecting a data set name and field, the translation operation would fail during actual reconciliation or provisioning operations.

      This point is also mentioned in the Mappings section .

      For the Country field example described earlier, select **Literal** and select the lookup definition you create in Step 1.

12.5.2.1 Configuring Account Status Reconciliation

User account status information is used to track whether or not the owner of a target system account is to be allowed to access and use the account. If required, you can use the translation transformation provider to reconcile account status information.

Note: The Design Console offers an alternative method to configure account status reconciliation. This method does not involve the use of a generic technology connector. Section 2.1.2.2.1, "User Account Status Reconciliation" describes this method.

You need to use the translation transformation provider only if account status values used in the target system are not the same as the values used in Oracle Identity Manager. For a target resource, Oracle Identity Manager uses the following values:

- Enabled state: `Enabled`
- Disabled state: `Disabled`

For a trusted source, Oracle Identity Manager uses the following values:

- Enabled state: `Active`
- Disabled state: `Disabled`

The procedure to configure account status reconciliation can be summarized as follows:

Note: Detailed instructions to perform these steps are provided later in this section.

---

1. Create a lookup definition that maps the status values used in the target system with the values used in Oracle Identity Manager.
2. While creating the generic technology connector, use the translation transformation provider to create a transformation mapping between the fields that hold account status values in the Source data set and the reconciliation staging data set.

   The following example describes the action that you must perform:

   Suppose the following fields are used to hold account status values:

   - The User Status field of the Source data set holds the values `True` (for a user in the Enabled state) and `False` (for a user in the Disabled state).
   - The User Status field of the reconciliation staging data set must hold one of the following pairs of values:
     - For target resource reconciliation, the field must hold `Enabled` or `Disabled`.
     - For trusted source reconciliation, the field must hold `Active` or `Disabled`.

   You must create a transformation mapping that converts the `True/False` values in the User Status field of the Source data set into corresponding `Enabled/Disabled` or `Active/Disabled` values. During reconciliation, these converted values are sent to the User Status field of the reconciliation staging data set.
3. Create a mapping between the field that holds account status values in the reconciliation staging data set and one of the following fields:
   - The OIM Object Status field of the OIM – Account data set, for target resource reconciliation
   - The Status field of the OIM – User data set, for trusted source reconciliation

During reconciliation, this mapping is used to propagate status values from the reconciliation staging data set to the OIM – Account or OIM – User data set.

Detailed steps to configure account status reconciliation are as follows:

1. Create a lookup definition that maps the status values used in the target system with the values used in Oracle Identity Manager.

   The Code Key values in the lookup definition must be the same as the values used to represent the account status in the target system. The Code Key and Decode values for both trusted and target resource reconciliation are as shown in the following table:

Code Key	Decode (for Trusted Source Reconciliation)	Decode (for Target Resource Reconciliation)
Target system status value for a user account that is in the Enabled state	`Active`	`Enabled`
Target system status value for a user account that is in the Disabled state	`Disabled`	`Disabled`

   Examples of Code Key values are `True/False`, `Yes/No`, and `1/0`. The Decode values must be set to the exact value, including the case (uppercase and lowercase), shown in the table.

   Note: While creating the lookup definition in the Lookup Definition form, you must select the Lookup Type option, and not the Field Type option.

2. The procedure to create the generic technology connector is described in Chapter 11, "Using Generic Technology Connectors". While creating the generic technology connector, perform the following steps on the Step 3: Modify Connector Configuration page:

   Note: These steps are a condensed version of the procedure described in "Adding or Editing Fields in Data Sets" in *Administering Oracle Identity Manager*. Refer to that section for a description of the terms and GUI elements mentioned in the following steps.

   a. If the target system status field is displayed on the Step 3: Modify Connector Configuration page, click the Edit icon for the field in the reconciliation staging data set.

      If the field is not displayed, click the Add icon of the reconciliation staging data set.

   b. On the Step 1: Field Information page, specify values for the following GUI elements:

      – **Field Name**: If you are adding the field, specify a name for it. The field name that you specify must contain only ASCII characters, because non-ASCII characters are not allowed.

      – Mapping Action: Select **Create Mapping With Translation** from this list.

- **Matching Only**: Ensure that this check box is deselected.
- **Create End-to-End Mapping**: If you are adding the field, select this check box.
- **Multi-Valued Field**: Ensure that this check box is deselected.
- **Data Type**: Select the data type of the field.
- **Length**: Specify the character length of the field.
- **Required**: Select this check box if you want to ensure that the field always contains a value.
- **Encrypted**: Ensure that this check box is deselected.
- **Password Field**: Ensure that this check box is deselected.

c. Click **Continue**.

d. On the Step 3: Provide Mapping Information page, perform the following steps:

In the Input region:

- From the **Dataset** list, select **Source**.
- From the **Field Name** list, select the field that stores status values.

In the Lookup Code Name region, select **Literal** and enter the name of the lookup definition that you create in Step 1.

e. If required, select a validation check for the field and click **Add**. In other words, select the validation provider that you want to use.

f. Click **Continue**, and click **Close**.

3. Create a mapping between the status field of the reconciliation staging data set and either the OIM Object Status field of the OIM - Account data set or the Status field of the OIM - User data set as follows:

Note: These steps are a condensed version of the procedure described in "Adding or Editing Fields in Data Sets" in *Administering Oracle Identity Manager*.

a. For target resource reconciliation, click the edit icon for the OIM Object Status field of the OIM - Account data set.

For target resource reconciliation, click the edit icon for the Status field of the OIM - User data set.

Note: If a mapping already exists between the status field of the reconciliation staging data set and the OIM Object Status field or Status field, apply the instructions given in this step only where required.

b. On the Step 1: Field Information page, specify values for the following GUI elements:

- Mapping Action: Select **Create Mapping Without Transformation** from this list.

- **Matching Only**: Ensure that this check box is deselected.

c. Click **Continue**.

d. In the Input region on the Step 3: Mapping page, select the status field of the reconciliation staging data set.

e. Click **Continue**, **Continue**, and click **Close**.

f. To add or edit other fields displayed on the Step 3: Modify Connector Configuration page, continue with the procedure described in "Adding or Editing Fields in Data Sets" in *Administering Oracle Identity Manager*.

12.6 Validation Providers

Table 12–1 describes the validation providers that are shipped with Oracle Identity Manager.

Note: Except for the Validate Date Format provider, all the providers in this table are implementations of methods of the `GenericValidator` class in the Apache Jakarta Commons API.

Table 12–1 Validation Providers

Validation Provider	Description
IsBlankOrNull	Returns true if the field value is null and is not blank
IsInRange	Returns true if the field value is within a range specified by a minimum and maximum value pair
IsByte	Checks if the field value can be converted to a byte primitive
IsDouble	Checks if the field value can be converted to a double primitive
IsFloat	Checks if the field value can be converted to a float primitive
IsInteger	Checks if the field value can be converted to an integer primitive
IsLong	Checks if the field value can be converted to a long primitive
IsShort	Checks if the field value can be converted to a short primitive
MatchRegexp	Checks if the field value matches the specified regular expression **Note:** A regular expression is a string that is used to describe or match a set of strings according to specific syntax rules.
MaxLength	Checks if the length of the field value is less than or equal to the specified value
MinLength	Checks if the length of the field value is greater than or equal to the specified value
Validate Date Format	Validates date values in target system records before these records are reconciled into Oracle Identity Manager The value of the Source Date Format parameter is used as the basis for validation. This validation provider is applied if you specify a value for the Source Date Format parameter on the Step 2: Specify Parameter Values page, regardless of whether or not you select this provider on the Step 3: Modify Connector Configuration page. **Note:** Unlike the other providers in this table, the Validate Date Format is not an implementation of a method of the `GenericValidator` class in the Apache Jakarta Commons API.

Part III

Workflows

This part describes how to develop workflows for customizing requests and approval processes, certification, and identity audit.

It contains the following chapter:

13 Developing Workflows

This chapter describes the concepts, features, and architecture of workflows in Oracle Identity Manager. It provides use cases for workflow, and instructions for designing, implementing, and deploying your first workflow. In addition, this chapter describes how to extend the request management operations by using plug-in points and how to customize the certification and identity audit composites.

This chapter contains the following sections:

13.1 Introducing Workflows

This section describes the key workflow concepts in the following sections:

13.1.1 Overview of Workflows

Managing user access and orchestrating the business process so that users get the correct access is a key identity governance function. The process of changing users' access can be initiated by the users through events in HR that trigger policies, or by administrators. Irrespective of how the change in access is initiated, organizations require the following:

- The business process that is initiated must be flexible, and must be able to meet changing business rules of the organization.
- The business process must be able to decide between granting access immediately versus introducing manual intervention steps and seeking approval prior to granting access.
- The business process must be able to perform validations, including Segregation of Duties (SoD) checks on what is being requested, by who, for whom, and in what context.
- If manual intervention is required, then the business process must have the ability to assign to users or groups of users and escalate, reassign, or expire if no response is received in a timely manner.
- For manual intervention, the business process must have the ability to gather information from the approvers, including comments and attachments.
- The business process must be able to interact with external systems, such as ticketing systems, when automated access grants are not possible, or the organization's rules require that access is granted manually.
- All decisions and actions must be audited and available in a reportable manner to allow the organization to measure performance of the process and also for auditors to fulfill compliance requirements

Oracle Identity Manager provides flexible and powerful access request capabilities that allow organizations to meet these requirements.

13.1.2 Workflow Concepts

The key concept of workflows in Oracle Identity Manager involves the following terminologies:

- **Request**

  In Oracle Identity Manager, a request refers to the business process that is invoked when an operation on an identity or an account has to be performed. Examples of these operations include creating a user, provisioning an account, and granting a role to a user. A request can either be fulfilled immediately (also known as direct operation) or can require manual intervention in the form of approvals (also known as request-based operation). When a user tries to perform an operation, Oracle Identity Manager determines whether the operation would be direct or request-based on the authorization policies of the logged-in user.

- **Approval**

  A request goes through one level of approval. This is configured in the workflow policy of the corresponding operation.

  A bulk requests goes through one level of approval. This is configured in the workflow policy of the corresponding Bulk operation. After approval, child requests are generated. These child requests would follow same approval process as simple request.

- **Approval workflow policy**

  An approval workflow policy consist of a rule configured by the administrator that allows the request engine to pick a SOA composite to invoke. The rules defined in approval workflow policies help the request engine determine if the request should be auto-approved or a SOA composite should be invoked.

- **SOA composite**

A SOA composite is an assembly of services, service components, and references designed and deployed together in a single application. Wiring between the service, service component, and reference enables message communication. The composite processes the information described in the messages.

- **Partner Link**

  A partner link enables you to define the external services with which the BPEL process service component is to interact. You can define partner links as services or references (for example, through a JCA adapter) in the SOA Composite Editor or within a BPEL process service component in Oracle BPEL Designer.

- **BPEL process**

  BPEL processes provide process orchestration and storage of synchronous or asynchronous processes. You design a business process that integrates a series of business activities and services into an end-to-end process flow.

- **IT provisioner**

  The IT provisioner, also known as fulfillment user or Help Desk user, is the persona responsible for fulfilling manual provisioning requests.

- **Request web service**

  The request web service is a web service that is shipped with Oracle Identity Manager. It allows customers to expose request, user, role, organization, account, entitlement, application instance, and catalog information so that approval workflows can make data-driven routing decisions.

- **Request callback**

  The request callback is a web service that is invoked by the SOA composite when an approval outcome (approve/ reject) has been received. When the request engine invokes a SOA composite for the purpose of approval, it suspends the request until the composite invokes the request callback and provides an approve or reject decision. This decision allows the request engine to proceed with fulfilling the request (if approved) or rejecting the request (if rejected).

- **Provisioning callback**

  The provisioning callback is a web service that is invoked as part of disconnected provisioning. When the IT provisioner or fulfillment user fulfills a disconnected provisioning request and marks the task as completed, the SOA composite invokes the provisioning callback and sends the provisioning status allowing the provisioning workflow to complete.

- **Request payload**

  The request engine invokes the SOA composite and passes it some basic information about the request, requester, and target user. This information is called the request payload.

- **Human Task**

  Human tasks provide workflow modeling that describes the tasks for users or groups to perform as part of an end-to-end business process flow.

13.1.3 Workflow Architecture

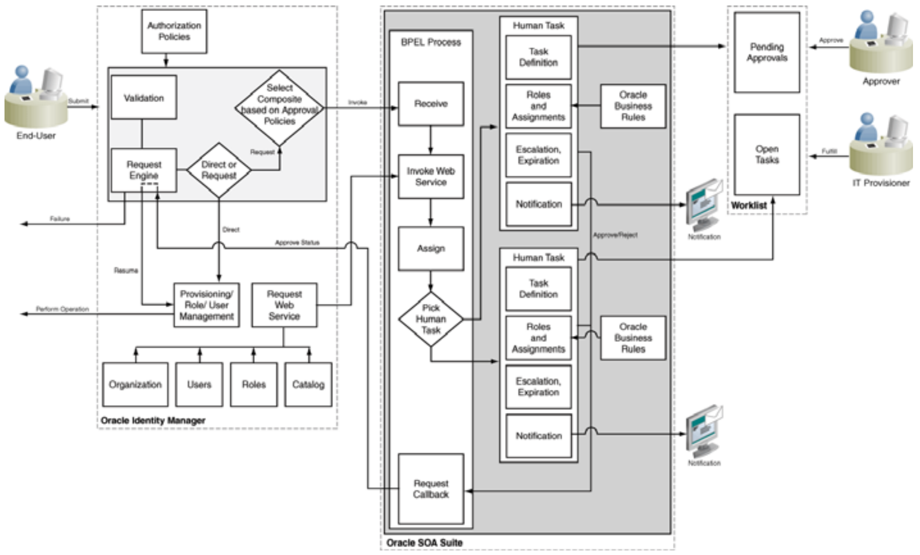
Workflows are used in Oracle Identity Manager to:

- Route requests to approvers for approval

- Route manual provisioning tasks to IT provisioners or Help Desk for fulfillment

Figure 13–1 provides an overview of workflows in Oracle Identity Manager:

Figure 13–1 Workflow Architecture



In Figure 13–1, the following actions occur:

1. User initiates an operation that results in a request. Examples of such operations include:
   - Self-registration
   - User profile modification, excluding lock, unlock, and password management operations
   - Role grant operations
   - Application instance operations, including disconnected provisioning
   - Entitlement operations
   - Bulk operations
2. A request is created. After appropriate validation, the request engine evaluates approval workflow policies and selects a SOA composite to be invoked.
3. If approval workflow policies are not configured, then the default SOA composite is selected for approval.
4. The SOA composite involves the Business Process Execution Language (BPEL) process.
5. The BPEL process invokes a web service to get additional details about the request including:

   Note: This step is optional. This is required only if additional information related to various entities is required in BPEL Process.

   - Item details from the catalog
   - Target user information

- Requester information

6. The BPEL process invokes additional logic to calculate properties such as priority, approvers, and notification.
7. When manual intervention is required, such as during approval and manual fulfillment, the process invokes a Human Task.

   A Human Task contains the logic to assign, expire, or escalate the approval task to users or roles. The Human Task can assign the users and roles statically or dynamically. For static assignments, the approvers can be determined in the BPEL process and passed as parameters to the Human Task. For dynamic assignments, rules created using Oracle Business Rules (OBR) are used to dynamically determine the approvers.

   Typically, the BPEL process contains one Human Task. In some instances, the BPEL process might invoke a decision point to pick one of multiple Human Tasks.
8. When the human task completes, a response of approve or reject (for approval) or complete (for manual fulfillment), is returned via a callback service to Oracle Identity Manager, which resumes the operation.

13.2 Predefined SOA Composites

Table 13–1 lists the predefined SOA composites in Oracle Identity Manager that can be used as approval processes.

Table 13–1 Predefined Workflow Composites

Workflow Composite	Description
DefaultRequestApproval	This is the default request-level approval. By default, the request-level approval goes to the SYSTEM ADMINISTRATORS role, for request-level approval. In addition, this composite is invoked by certification use cases. The task will have one of the following states: ■ Assigned to the beneficiary. Later, the task may be assigned to the beneficiary's manager based on the decision of the beneficiary. ■ Auto-approved if the certification requester is beneficiary's manager. **Note:** For information about the certification use cases, see "Managing Identity Certification" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager*.
DefaultOperationalApprova l	This is the default operation-level approval. By default, the approval task is assigned to the SYSTEM ADMINISTRATORS role for operation-level approval. In addition, the composite is invoked by certification use cases, and the task will be auto-approved.
BeneficiaryManagerApprov al	This requires approval from the beneficiary's manager. This can be associated with the following: ■ The request models that have a beneficiary. Examples of such request models are Provision Application Instance and Assign Roles. ■ All user models except Create User and Self-Register User. This composite must be associated at the operational level of approval because a request can have multiple beneficiaries at the request level.

Table 13–1 (Cont.) Predefined Workflow Composites

Workflow Composite	Description
DefaultRoleApproval	This SOA composite creates a single approval task that is assigned to the SYSTEM ADMINISTRATORS role for approval.
RequesterManagerApproval	This SOA composite creates a single approval task that is assigned to the requester's manager for approval. **Note:** This composite cannot be associated with unauthenticated request models, such as Self-Register User.
DefaultSODApproval	This SOA composite creates an approval task that is assigned to the System Administrator, starts SoD check, and after the SoD result is available, it creates another approval task assigned to the SOD Administrators role. This must be associated with request models to provision or modify resources at the operational level if SoD check is required.
DisconnectedProvisioning	This SOA composite assigns the task to the System Administrator to fulfil the disconnected provisioning.
ProvideInformation	This SOA composite assigns the task to the requester seeking details of account/entitlement.
CertificationProcess	This is the default Certification composite. This composite takes care of assigning the certification task to the certifier (user). This composite also manages the following certification task events: ■ Expiry ■ Proxy ■ Escalation ■ Re-assignment
CertificationOverseerProcess	This composite assigns a certification task to the certifier (user). In addition, the composite also handles routing the task to the overseer after the certifier completes the task. Oracle SOA Business Rules are used to handle the task routing. This composite handles the following certification task events: ■ Expiry ■ Proxy ■ Routing (Overseer) ■ Escalation ■ Re-assignment

13.3 Creating New SOA Composites

To create a new SOA composite that can be used as an approval process, perform the following steps:

1. Creating a New SOA Composite
2. Deploying a SOA Composite in Oracle SOA Server
3. Prerequisites for Communication to Oracle Identity Manager Through SSL Mode

13.3.1 Creating a New SOA Composite

To use a SOA composite as an approval process, it must adhere to certain standards. These standards ensure that the request service is able to instantiate and manage such composites correctly. These standards are:

- The following attributes are mandatory for BPEL process:

  - RequestID of type String
  - RequestModel of type String
  - RequestTarget of type String
  - URL of type String
  - RequesterDetails of XML Element
  - BeneficiaryDetails of XML Element
  - ObjectDetails of XML Element
  - OtherDetails of XML Element

  The RequestID, RequestModel, RequestTarget, and URL attributes are always set with valid values for all types of requests.

  RequesterDetails is an XML element. This element is filled up with valid values for all requests that requires authentication. Requester details is empty for the requests of type Self-Register User because the requester is anonymous user.

  BeneficiaryDetails is an XML element. This element is filled up with valid values for all requests that have a beneficiary, for example, Provision Resource and Assign Roles. This is filled up only if the request is associated with single beneficiary. If the request is associated with multiple beneficiaries, then BeneficiaryDetails is empty. BeneficiaryDetails element always has valid value for simple requests and child requests that have a beneficiary. Therefore, it is recommended to use this XML element in SOA composites that are used as approval processes at the operational level of approval. This is because at the operational level of approval, the request is associated with only one beneficiary.

  ObjectDetails is an XML element. This element is filled up with valid values for all requests that are associated with the Resource entity. This is filled up only if the request is associated with single resource. If the request is associated with multiple resources, then ObjectDetails is empty. The ObjectDetails element always has valid value for simple and child requests that are associated with resource. Therefore, it is recommended to use this XML element in SOA composites that are used as approval processes at the operational level of approval. This is because at the operational level of approval, the request is associated with only one resource.

- All the attributes that are mandatory for the BPEL process are referred from RequestDetails.xsd and ApprovalProcess.xsd. These files are present in the template SOA composite, which must not de modified or deleted.

Oracle Identity Manager provides a helper utility for creating custom SOA composites. This utility creates a template SOA project that adheres to all the necessary standards. This utility is located in the *OIM_HOME*/workflows/new-workflow directory.

Note:

- JAVA_HOME environment variable must be set before running this utility.
- This utility requires Apache Ant version 1.7 or later.

To create a custom SOA composite by running the helper utility:

1. Run the following commands:

```
cd OIM_HOME/workflows/new-workflow
ant -f new_project.xml
```

2. Enter the JDeveloper application name when the following prompt is displayed:

   ```
   Please enter application name
   ```

3. Enter the JDeveloper project name when the following prompt is displayed:

   ```
   Please enter project name
   ```

4. Enter the name of the ADF binding service for the composite when the following prompt is displayed:

   ```
   Please enter the service name for the composite. This needs to be
   unique across applications
   ```

The new application is created in the *OIM_HOME*/workflows/new-workflow/process-template/ directory. You can open the new application in JDeveloper for modification.

Human task in the template SOA composite is configured to send notifications to the assignee of the human task. In the custom composite that is created, the notification message can be modified based on the requirement. All the notifications to be sent to the approver must be configured in the SOA composite. For configuring Oracle SOA server to send notifications, refer to "Configuring Oracle User Messaging Service" in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*.

Human task in the template SOA composite is configured to be assigned to the SYSTEM ADMINISTRATORS role.

13.3.2 Deploying a SOA Composite in Oracle SOA Server

For information about deploying the workflow composite in BPEL, see *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

Note: The composite should be redeployed with a new version. If a composite is redeployed with the same version in SOA, then all the pending approvals in Oracle Identity Manager initiated by the composite becomes stale and are removed from the user's TaskList. See "Deploying an Existing SOA Archive in Oracle JDeveloper" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite* for information about deploying existing SOA composites.

13.3.3 Prerequisites for Communication to Oracle Identity Manager Through SSL Mode

If the communication to Oracle Identity Manager is through the SSL mode, then you must:

Note: For a non-SSL connection, skip this section.

- Set the *TRUSTSTORE_LOCATION* environment variable, where *TRUSTSTORE_LOCATION* is the trusted key store file location.
- Use t3s protocol instead of t3. For example, the URL for Oracle Identity Manager is:

  t3s://*HOST_NAME*:*PORT*

13.4 Developing Workflows: Vision Request Tutorial

This section describes how to design your first workflow in the following sections:

- Introducing the Tutorial
- Prerequisites
- Creating the Application Instance
- Configuring FinApp in the Catalog
- Creating and Configuring the SOA Composite for Approval

13.4.1 Introducing the Tutorial

This tutorial is based on the following use case:

- Vision Corp uses FinApp, a mainframe-based application. The application does not have APIs that can be remotely invoked. Therefore, accounts are managed manually by the Help Desk.
- Vision Corp employees use the Access Request Catalog to request accounts and entitlements in the application.
- Approvals are based on the risk level of the access being requested. If the risk level is Low Risk, approval is required only from the beneficiary's manager. If the risk level is Medium Risk, approval is required from either the beneficiary's manager or certifier. If the risk level is High Risk, approval is required from the beneficiary's manager and the Audit Review team.
- After approval, the request has to be fulfilled by members of the Asset Management team.

This tutorial describes how to create the application and the workflow, and how to configure the approval and fulfillment for the application.

The result of the tutorial is:

- An application instance
- A SOA composite for approval consisting of:
  - A BPEL process
  - Multiple Human Tasks
- Modification to the predefined Disconnected Provisioning SOA composite

13.4.2 Prerequisites

The following assumptions are made for this tutorial:

- Oracle SOA Suite is installed on a host on which the SOA infrastructure is configured.
- JDeveloper 11.1.1.9 with SOA Design Time 11.1.1.9 is available.
- Mandatory patches, if any, for SOA Jdeveloper extension have been applied. For information about the mandatory patches, see "Mandatory Patches Required for Installing Oracle Identity Manager" in the *Oracle Fusion Middleware Release Notes*.
- You are familiar with basic BPEL constructs, including BPEL activities and partner links, and basic XPath functions.

- You are familiar with the SOA Composite Editor and Oracle BPEL Designer, the environment for designing and deploying BPEL processes. However, for detailed information about SOA composites, refer to *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.
- Two roles, Audit Review Team and Asset Management Team, have been created and members have been assigned.
- An organization with name Vision is created.

13.4.3 Creating the Application Instance

This section contains the following topics:

- Creating the FinApp Application Instance
- Defining Application Instance Attributes and Creating a Form
- Publishing the Application Instance to One or More Organizations
- Linking Entitlements to the Application Instance
- Publishing the Application Instance With Entitlements to the Catalog

13.4.3.1 Creating the FinApp Application Instance

To create the FinApp application instance:

1. Login to Oracle Identity System Administration.
2. Click Sandboxes to access sandbox management, create a sandbox, and activate it. See "Managing Sandboxes" on page 19-1 for information about sandboxes and how to create, activate, and publish sandboxes.
3. Under Configuration, click **Application Instances**. Click **Create** on the toolbar to open the Create Application Instance page.
4. Enter Name and Display Name as **FinApp**.
5. Select the **Disconnected** option to specify a disconnected application instance.
6. Click **Save**, and then click **OK** to confirm creation of the FinApp application instance.

13.4.3.2 Defining Application Instance Attributes and Creating a Form

To define the attributes of the application instance and create a form:

1. Under Configuration, click **Form Designer**.
2. Search and select the FinApp form. This form is automatically created when the disconnected application instance is saved.

Note: You must be in an active sandbox to create and edit a form.

3. Click the **Fields** tab, and then click the Edit icon on the toolbar.

   By default, the following fields are created and are available for use:

Field	Description
IT Resource	The IT resource instance where the account is being created

Field	Description
Account Login	The login for the application
Password	The password that is used while logging in to the application
Account ID	The unique identifier generated by the application when the account is created
Service Account	A flag that is used during access request only

Note: Attributes such as Account ID and IT Resource are typically not displayed in the access request user interface. Depending upon the use case, for example a mobile phone request, the attributes might not be relevant. To hide these attributes, you can customize the form. See "Configuring Custom Attributes" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager* for more information on how to customize the form.

4. Add additional attributes. In this example, add the following attribute:

   Account Description: Data type is Text.

   Note: See "Configuring Custom Attributes" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager* for more information on creating the custom attributes

5. After adding the attributes, verify that the configuration in the Fields tab is similar to the following table:

Display Label	Name	Type
Account Description	AccountDescription	Text
Account ID	UD_FINAPP_ACCOUNTID	Text
ITResource	UD_FINAPP_IT	Number
Account Login	UD_FINAPP_LOGIN	Text
Password	UD_FINAPP_PASSWORD	Text
Service Account	serviceaccount	Checkbox

6. To allow users to request entitlements, you must add a child object and add an attribute that is tagged as an Entitlement. To do so:
   a. Click the **Child Objects** tab, and then click **Add** on the toolbar.
   b. Enter the child object name, and click **OK** to create the child object.
   c. Click the child object just created.
   d. Select **Action**, **Create** to create a new attribute. From the popup window, select **Lookup**, and click **OK**. Enter values for the following fields:

      Name: Profile Name

      Display Name: Profile Name

e. Select **Use in Bulk** to allow requesters to specify a value when requesting access for multiple users.

f. Under Lookup Type, click **Create a New Lookup Type**.

g. Create the new Lookup and specify the values, as shown:

   Code: Lookup.FinApp.Profile

   Meaning: Lookup.FinApp.Profile

   Description: Lookup.FinApp.Profile

h. Create three lookup codes by using the values given in the following table:

Meaning	Code
FinApp User	FinAppUser
FinApp Administrator	FinAppAdministrator
FinApp Operator	FinAppOperator

Note: You can also populate the lookup definition by using a scheduled task and the lookup APIs.

i. Select the **Searchable**, **Entitlement**, and **Searchable Picklist** options.

7. Click **Save and Close**.
8. Click **Back to Parent Object**.
9. Click **Regenerate View** to re-create the UI form with the new attributes.
10. Close all tabs.
11. Publish the sandbox.

13.4.3.3 Publishing the Application Instance to One or More Organizations

To publish the application instance to one or more organizations:

1. Open the FinApp application instance details page, and click the Organizations tab.
2. Click **Assign**. In the Select Organizations dialog box, select one or more organizations to publish the application instance to.
3. Select the **Hierarchy** option if you want the application instance to be published to the organization and its child organizations.
4. Click **OK**.

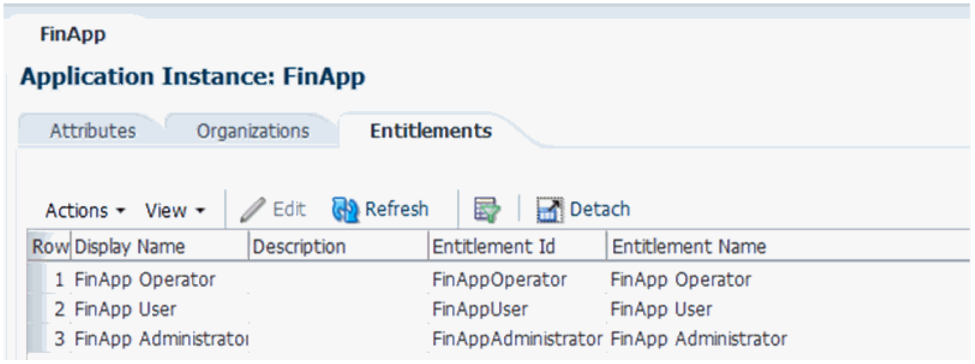
13.4.3.4 Linking Entitlements to the Application Instance

To link entitlements to the application instance:

1. Under System Management, click **Scheduler**.
2. Search for the Entitlement List scheduled job, and click **Run Now**.
3. Under Configuration, click **Application Instances**, and navigate to the FinApp application instance.

4. Click the **Entitlements** tab, and verify that the entitlements are displayed, as shown in Figure 13–2:

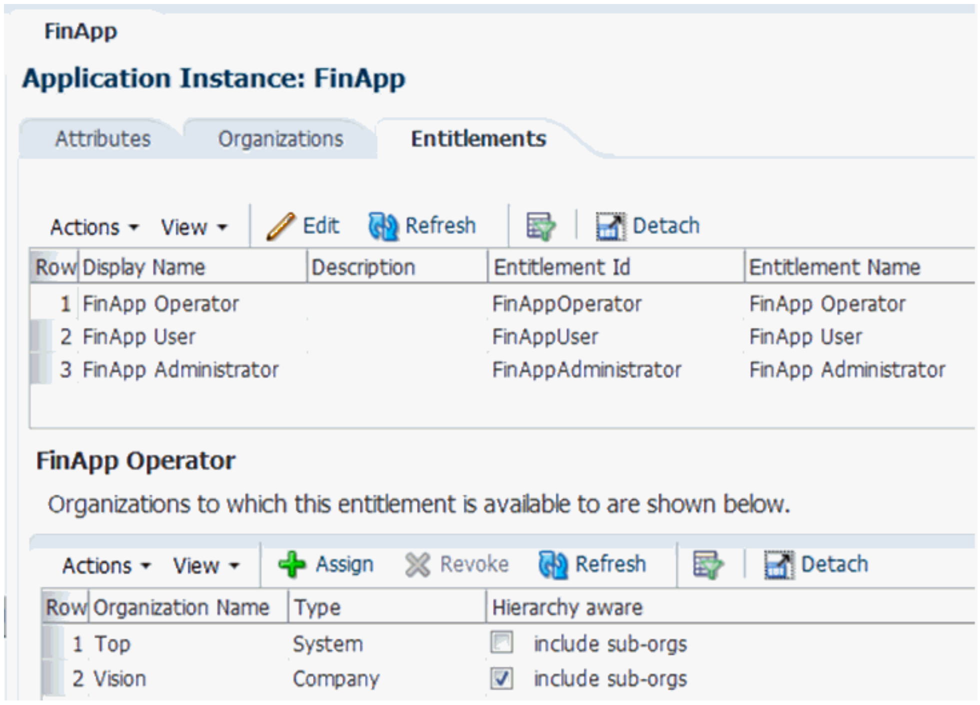
Figure 13–2 Entitlements List



Row	Display Name	Description	Entitlement Id	Entitlement Name
1	FinApp Operator		FinAppOperator	FinApp Operator
2	FinApp User		FinAppUser	FinApp User
3	FinApp Administrator		FinAppAdministrator	FinApp Administrator

5. Select an entitlement, and verify that it is published to the same organizations as the application instance, as shown in Figure 13–3:

Figure 13–3 Entitlement Availability to Organizations



Row	Display Name	Description	Entitlement Id	Entitlement Name
1	FinApp Operator		FinAppOperator	FinApp Operator
2	FinApp User		FinAppUser	FinApp User
3	FinApp Administrator		FinAppAdministrator	FinApp Administrator

FinApp Operator

Organizations to which this entitlement is available to are shown below.

Row	Organization Name	Type	Hierarchy aware
1	Top	System	☐ include sub-orgs
2	Vision	Company	☑ include sub-orgs

6. Edit one or more entitlements, and enter a business friendly description. If required, modify the display name as well.

13.4.3.5 Publishing the Application Instance With Entitlements to the Catalog

To publish the application instance and its entitlements to the catalog:

1. Under System Management, click Scheduler.
2. Search for the Catalog Synchronization scheduled job, and click **Run Now**.

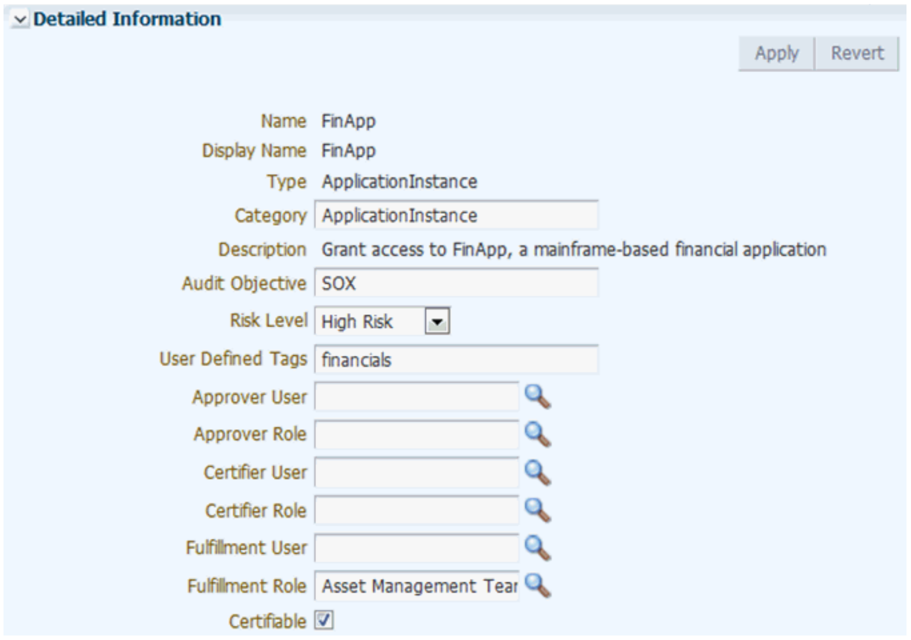
13.4.4 Configuring FinApp in the Catalog

To configure the application instance and its entitlements in the catalog:

1. Login to Oracle Identity Self Service as the Catalog Administrator.
2. Under Requests, click **Catalog**.
3. In the Catalog page, search for the application instance.
4. Select the application instance, and edit the catalog item details.
5. Provide values for the default attributes. Because this tutorial involves workflow routing based on risk level and manual fulfillment, you must provide a value for the Risk Level and Fulfillment Role attributes. However, it is recommended that you provide values for other attributes, especially User Defined Tags.

   Figure 13–4 shows the attributes of the catalog item.

Figure 13–4 Catalog Item Attributes



Name	FinApp
Display Name	FinApp
Type	ApplicationInstance
Category	ApplicationInstance
Description	Grant access to FinApp, a mainframe-based financial application
Audit Objective	SOX
Risk Level	High Risk
User Defined Tags	financials
Approver User	
Approver Role	
Certifier User	
Certifier Role	
Fulfillment User	
Fulfillment Role	Asset Management Tear
Certifiable	✓

13.4.5 Creating and Configuring the SOA Composite for Approval

This section contains the following topics:

- Creating the Approval Workflow
- Making Request and Catalog Data Available to the BPEL Process
- Configuring Workflow Selection
- Configuring Human Tasks
- Configuring the Human Task and BPEL Mappings
- Deploying the SOA Composite
- Creating the Workflow Rules

13.4.5.1 Creating the Approval Workflow

To create a new approval workflow:

1. Set the JAVA_HOME environment variable by running the setWLSEnv.sh script in the /server/bin/ subdirectory in the WebLogic Server installation directory.
2. Set the ANT_HOME environment variable to *MIDDLEWARE_HOME*/modules/org.apache.ant_1.7.1.
3. Set the PATH environment variable to $JAVA_HOME/bin:$ANT_HOME/bin:$PATH.
4. Navigate to *OIM_HOME*/server/workflows/new_workflow.
5. Run the following:

   ```
   ant-f new_project.xml
   ```

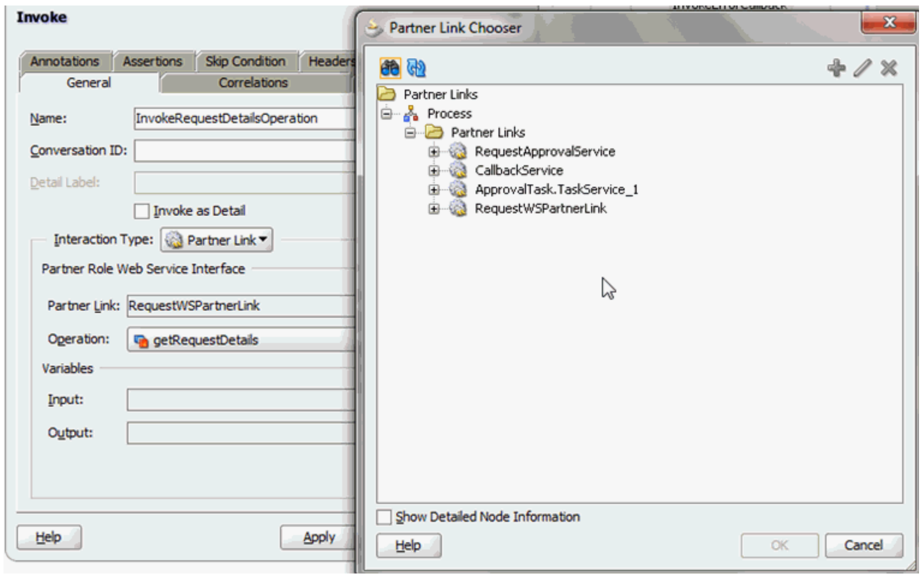
6. Provide the Application Name as AddAccessApprovalApplication.
7. Provide the Project Name as AddAccessApproval.
8. Provide the Service Name as AddAccess.
9. Wait for the utility to finish generating the new JDeveloper Workspace containing the Composite. The workspace is generated in /server/workflows/new-workflow/process-template.
10. Copy the directory to a location accessible to JDeveloper.

13.4.5.2 Making Request and Catalog Data Available to the BPEL Process

To make request and catalog data available to the BPEL process:

1. Switch to Design view of the BPEL process.
2. Drag the Invoke activity from the Component Palette and drop it below the AssignRequestWSURL activity. Rename it to `InvokeRequestDetailsOperation`.
3. Right-click **InvokeRequestDetailsOperation**, and select **Edit**.
4. Select partner link from the Partner Link Chooser as RequestWSPartnerLink, and operation as getRequestDetails, as shown in Figure 13–5.

Figure 13–5 Partner Link and Operation



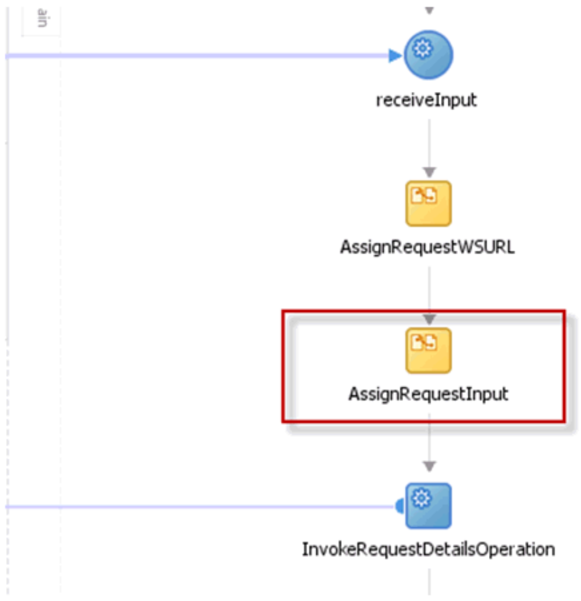
Name: InvokeRequestDetailsOperation
Conversation ID:
Detail Label:
Interaction Type: Partner Link
Partner Link: RequestWSPartnerLink
Operation: getRequestDetails
Input:
Output:

Partner Links
Process
Partner Links
RequestApprovalService
CallbackService
ApprovalTask.TaskService_1
RequestWSPartnerLink

5. Under the Variables section, click the plus (+) icon for the Input and Output fields to create the input and output variables. Name the input and output variables as `requestDetails_InputVariable` and `requestDetails_OutputVariable` respectively. Then click **Apply** and **OK**.

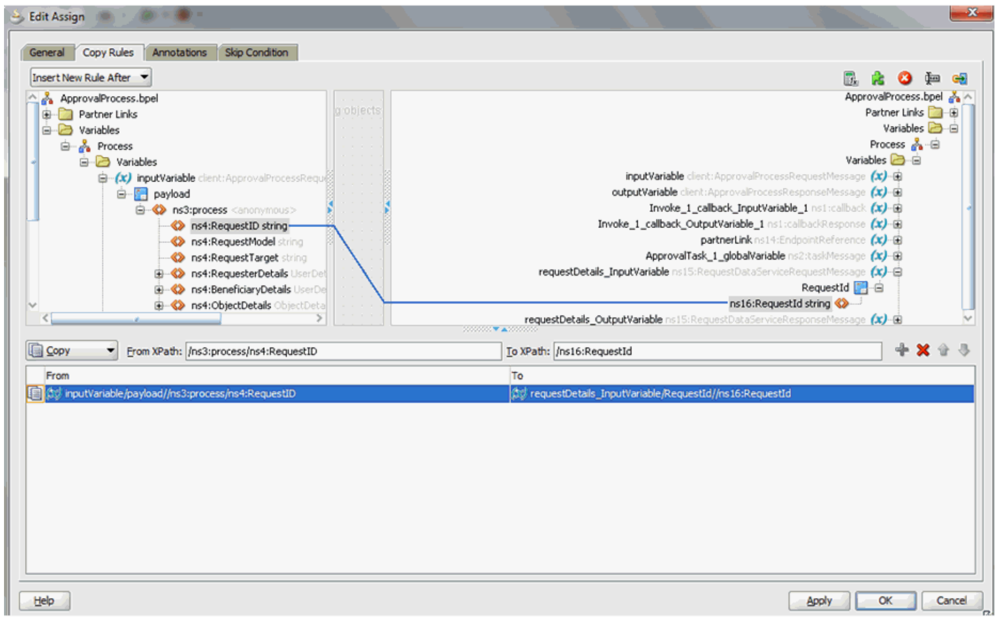
6. Drag and drop an assign activity, rename it to `AssignRequestInput`, and place it above the InvokeRequestDetailsOperation invoke activity, as shown in Figure 13–6.

Figure 13–6 AssignRequestInput



7. Right-click **AssignRequestInput**, and select **Edit** to map the input of the InvokeRequestDetailsOperation, as shown in Figure 13–7.

Figure 13–7 Input Mapping

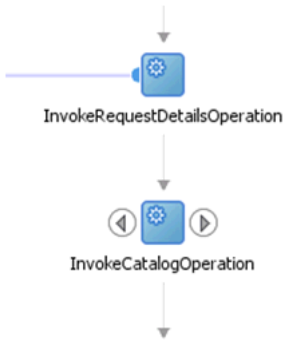

Edit Assign
General
Copy Rules
Annotations
Skip Condition
Insert New Rule After
ApprovalProcess.bpel
Partner Links
Variables
Process
Variables
inputVariable client:ApprovalProcessReque
payload
ns3:process <anonymous>
ns4:RequestID string
ns4:RequestModel string
ns4:RequestTarget string
ns4:RequesterDetails UserDet
ns4:BeneficiaryDetails UserDe
ns4:ObjectDetails ObjectDeta
ApprovalProcess.bpel
Partner Links
Variables
Process
Variables
inputVariable client:ApprovalProcessRequestMessage
outputVariable client:ApprovalProcessResponseMessage
Invoke_1_callback_InputVariable_1 ns1:callback
Invoke_1_callback_OutputVariable_1 ns1:callbackResponse
partnerLink ns14:EndpointReference
ApprovalTask_1_globalVariable ns2:taskMessage
requestDetails_InputVariable ns15:RequestDataServiceRequestMessage
RequestId
ns16:RequestId string
requestDetails_OutputVariable ns15:RequestDataServiceResponseMessage
Help
Apply
OK
Cancel

Copy | From XPath: /ns3:process/ns4:RequestID | To XPath: /ns16:RequestId

From	To
inputVariable/payload//ns3:process/ns4:RequestID	requestDetails_InputVariable/RequestId//ns16:RequestId

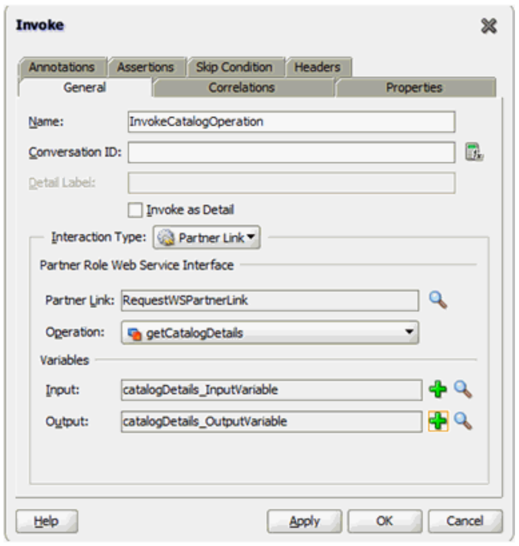
8. Add an Invoke activity after the InvokeRequestDetailsOperation, as shown in Figure 13–8. Name the activity `InvokeCatalogOperation`.

Figure 13–8 InvokeCatalogOperation



9. Edit the InvokeCatalogOperation, and configure it as shown in Figure 13–9.

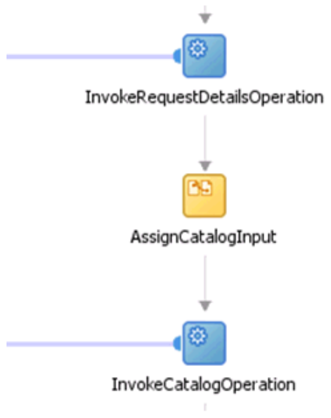
Figure 13–9 InvokeCatalogOperation Configuration



Name: InvokeCatalogOperation
Conversation ID:
Detail Label:
☐ Invoke as Detail
Interaction Type: Partner Link
Partner Role Web Service Interface
Partner Link: RequestWSPartnerLink
Operation: getCatalogDetails
Variables
Input: catalogDetails_InputVariable
Output: catalogDetails_OutputVariable

10. Add an Assign activity above InvokeCatalogOperation, as shown in Figure 13–10. Name the activity as AssignCatalogInput.

Figure 13–10 AssignCatalogInput



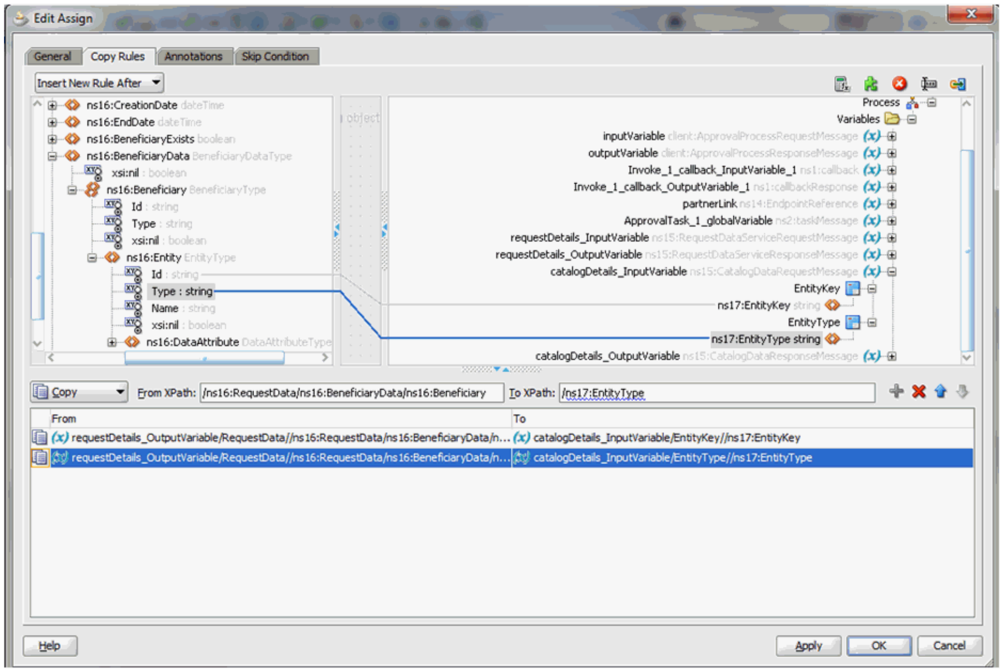
Note: The following attributes will be returned as custom attributes through the catalog detail method of the request web service:

- APPROVER_USER_FIRSTNAME
- APPROVER_USER_LASTNAME
- APPROVER_USER_DISPLAYNAME
- APPROVER_USER_EMAIL
- CERTIFIER_USER_FIRSTNAME
- CERTIFIER_USER_LASTNAME
- CERTIFIER_USER_DISPLAYNAME
- CERTIFIER_USER_EMAIL
- FULFILLMENT_USER_FIRSTNAME
- FULFILLMENT_USER_LASTNAME
- FULFILLMENT_USER_DISPLAYNAME
- FULFILLMENT_USER_EMAIL

---

11. Right-click and edit the assign activity to map the input to the InvokeCatalogOperation, as shown in Figure 13–11.

Figure 13–11 InvokeCatalogOperation Input Mapping



Copy From XPath: /ns16:RequestData/ns16:BeneficiaryData/ns16:Beneficiary To XPath: /ns17:EntityType

From	To
(x) requestDetails_OutputVariable/RequestData//ns16:RequestData/ns16:BeneficiaryData/n...	(x) catalogDetails_InputVariable/EntityKey//ns17:EntityKey
(x) requestDetails_OutputVariable/RequestData//ns16:RequestData/ns16:BeneficiaryData/n...	(x) catalogDetails_InputVariable/EntityType//ns17:EntityType

13.4.5.3 Configuring Workflow Selection

To define the workflow selection rules:

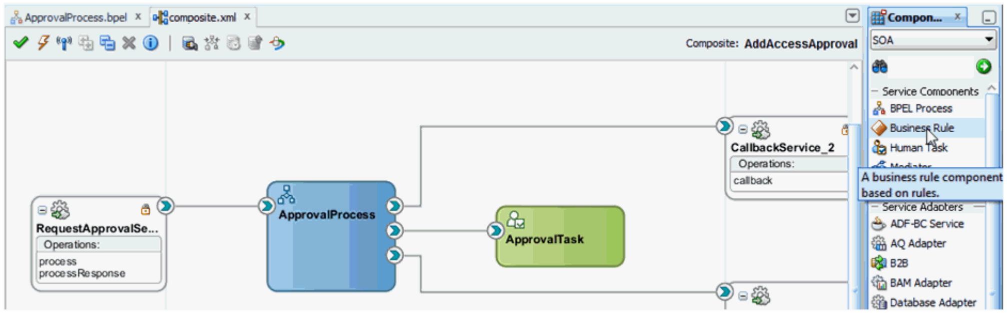
1. Define a variable called catalogData as Type - Element CatalogData. To do so, click the Variables icon, and then click the Create icon on the Variable dialog box.

   This variable will contain the catalog details returned as an output of the InvokeCatalogDetails step.

2. Define a variable called workflowtype as Type - Element StageOutput. This variable will contain the type of workflow to be invoked.

3. Navigate to the SOA Composite view, and add a Business Rule component, as shown in Figure 13–12.

Figure 13–12 Adding Business Rule Component



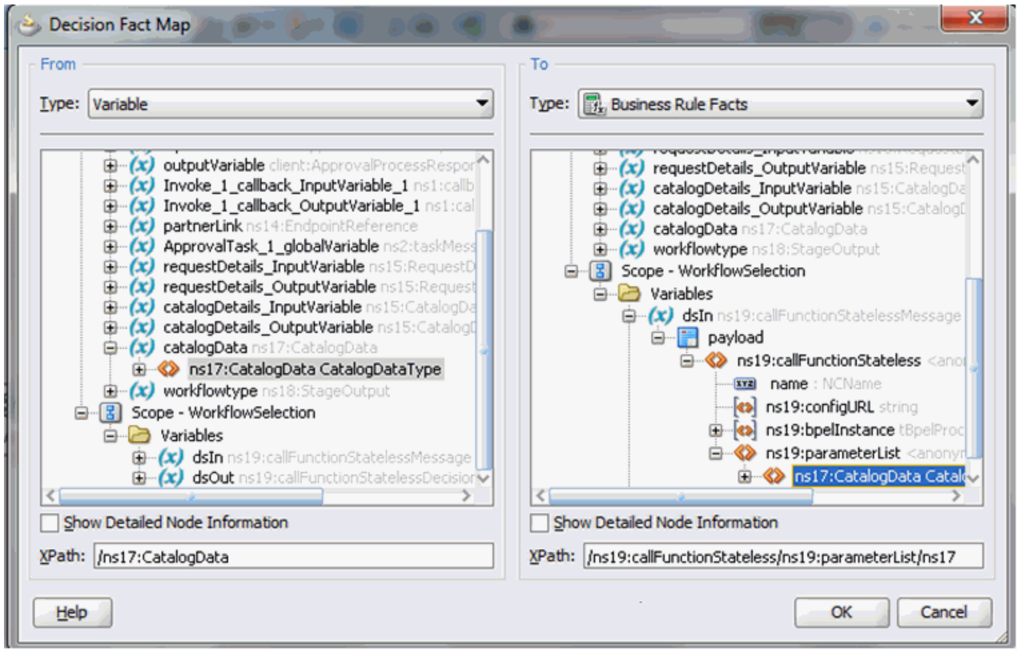
4. In the Create Business Rules dialog box, specify the name of the Rule Dictionary as WorkflowSelection.

5. Specify the Input as catalogData and the Output as workflowstage.

6. Switch to the BPEL process.

7. Expand SOA Components and add a Business Rule component.
8. Edit the rule and rename it to `WorkflowSelection`.
9. In the Rule dialog box, click the Dictionary tab, and select the WorkflowSelection dictionary that you defined in step 4.
10. Map the catalogData variable to the input to the Rule, as shown in Figure 13–13. Select Assign Input Facts subtab in the Dictionary tab, and click the plus (+) icon.
11. Map the catalogData variable to the input to the Rule, as shown in Figure 13–13.

Figure 13–13 catalogData Variable Input Mapping



Decision Fact Map

From

Type: Variable

```
outputVariable client:ApprovalProcessRespor
Invoke_1_callback_InputVariable_1 ns1:callb
Invoke_1_callback_OutputVariable_1 ns1:cal
partnerLink ns14:EndpointReference
ApprovalTask_1_globalVariable ns2:taskMess
requestDetails_InputVariable ns15:RequestD
requestDetails_OutputVariable ns15:Request
catalogDetails_InputVariable ns15:CatalogDa
catalogDetails_OutputVariable ns15:Catalog
catalogData ns17:CatalogData
    ns17:CatalogData CatalogDataType
workflowtype ns18:StageOutput
Scope - WorkflowSelection
  Variables
    dsIn ns19:callFunctionStatelessMessage
    dsOut ns19:callFunctionStatelessDecision
```

Show Detailed Node Information

XPath: /ns17:CatalogData

To

Type: Business Rule Facts

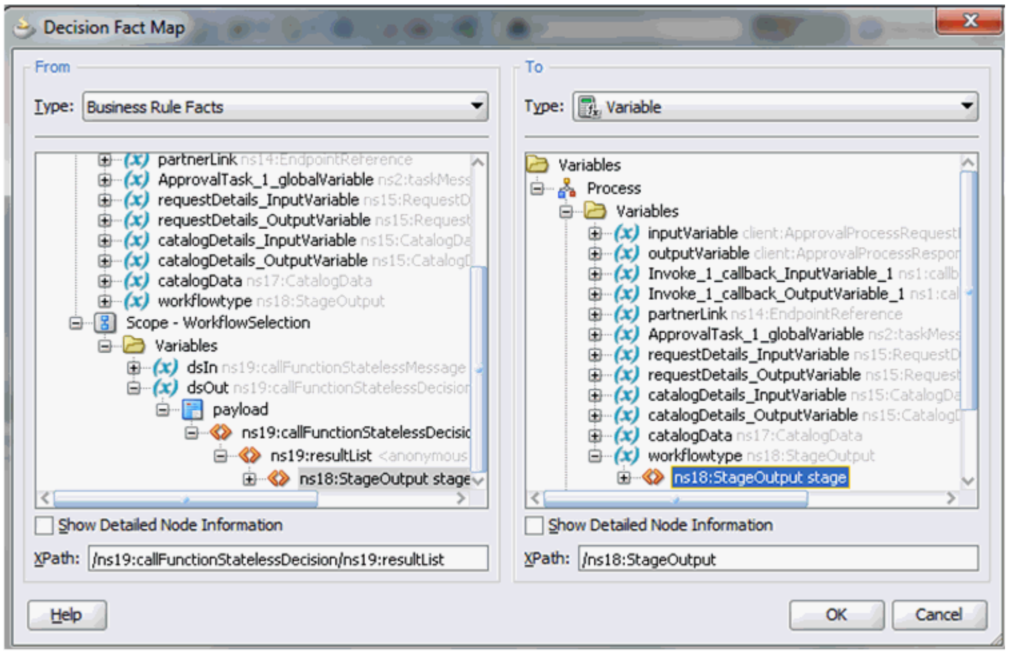
```
requestDetails_OutputVariable ns15:Request
catalogDetails_InputVariable ns15:CatalogDa
catalogDetails_OutputVariable ns15:Catalog
catalogData ns17:CatalogData
workflowtype ns18:StageOutput
Scope - WorkflowSelection
  Variables
    dsIn ns19:callFunctionStatelessMessage
      payload
        ns19:callFunctionStateless <ano
          name : NCName
          ns19:configURL string
          ns19:bpelInstance tBpelProc
          ns19:parameterList <anonyr
            ns17:CatalogData Catal
```

Show Detailed Node Information

XPath: /ns19:callFunctionStateless/ns19:parameterList/ns17

12. Map the workflowtype variable to the output to the Rule, as shown in Figure 13–14. Select Assign Output facts subtab in Dictionary tab.
13. Map the workflowtype variable to the output to the Rule, as shown in Figure 13–14.

Figure 13–14 workflowtype Variable Output Mapping



Decision Fact Map

From

Type: Business Rule Facts

- partnerLink ns14:EndpointReference
- ApprovalTask_1_globalVariable ns2:taskMess
- requestDetails_InputVariable ns15:RequestD
- requestDetails_OutputVariable ns15:Request
- catalogDetails_InputVariable ns15:CatalogDa
- catalogDetails_OutputVariable ns15:CatalogD
- catalogData ns17:CatalogData
- workflowtype ns18:StageOutput
- Scope - WorkflowSelection
  - Variables
    - dsIn ns19:callFunctionStatelessMessage
    - dsOut ns19:callFunctionStatelessDecision
      - payload
        - ns19:callFunctionStatelessDecisic
          - ns19:resultList <anonymous
            - ns18:StageOutput stage

Show Detailed Node Information

XPath: /ns19:callFunctionStatelessDecision/ns19:resultList

To

Type: Variable

- Variables
  - Process
    - Variables
      - inputVariable client:ApprovalProcessRequest
      - outputVariable client:ApprovalProcessRespor
      - Invoke_1_callback_InputVariable_1 ns1:callb
      - Invoke_1_callback_OutputVariable_1 ns1:cal
      - partnerLink ns14:EndpointReference
      - ApprovalTask_1_globalVariable ns2:taskMess
      - requestDetails_InputVariable ns15:RequestD
      - requestDetails_OutputVariable ns15:Request
      - catalogDetails_InputVariable ns15:CatalogDa
      - catalogDetails_OutputVariable ns15:CatalogD
      - catalogData ns17:CatalogData
      - workflowtype ns18:StageOutput
        - ns18:StageOutput stage

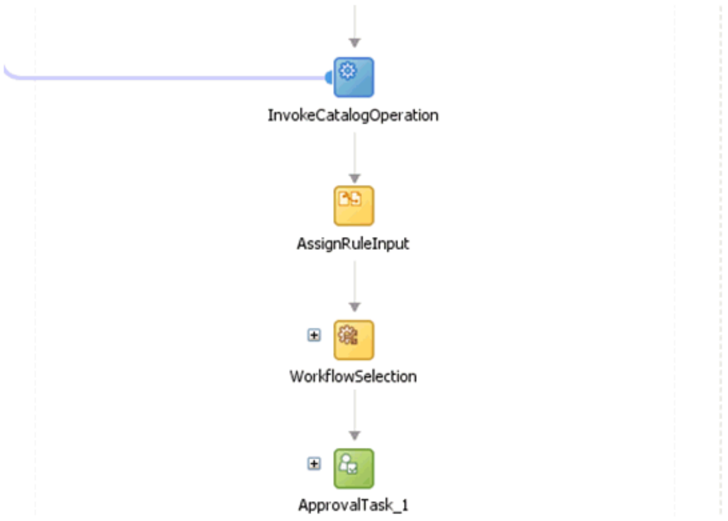
Show Detailed Node Information

XPath: /ns18:StageOutput

Help OK Cancel

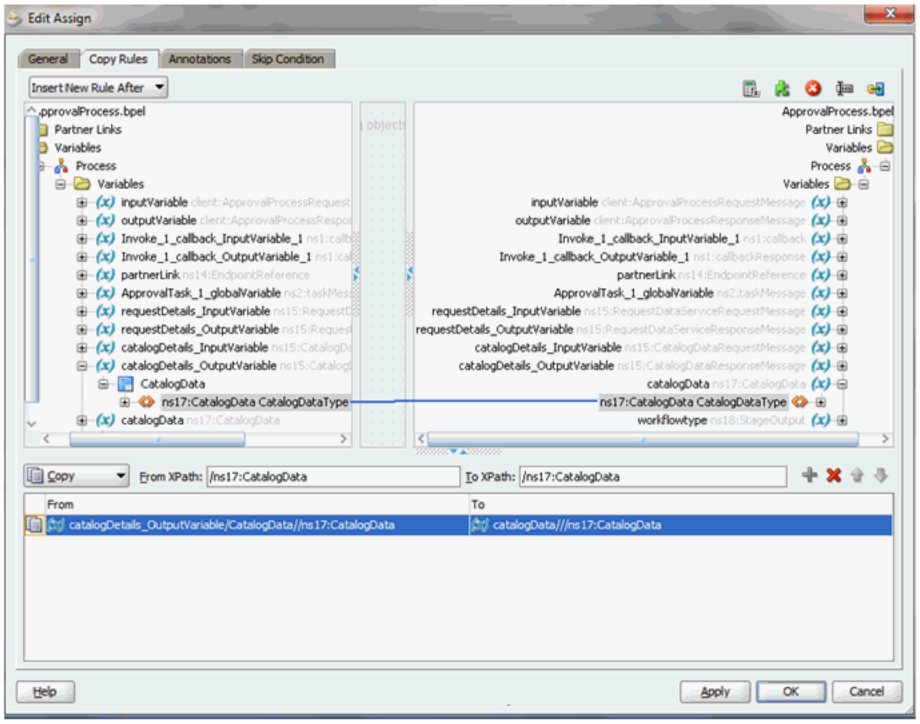
14. Add an Assign activity before the WorkflowSelection rule and rename it as AssignRuleInput, as shown in Figure 13–15.

Figure 13–15 AssignRuleInput



15. Map the output of the InvokeCatalogOperation to the catalogData variable, as shown in Figure 13–16.

Figure 13–16 catalogData Variable Output Mapping

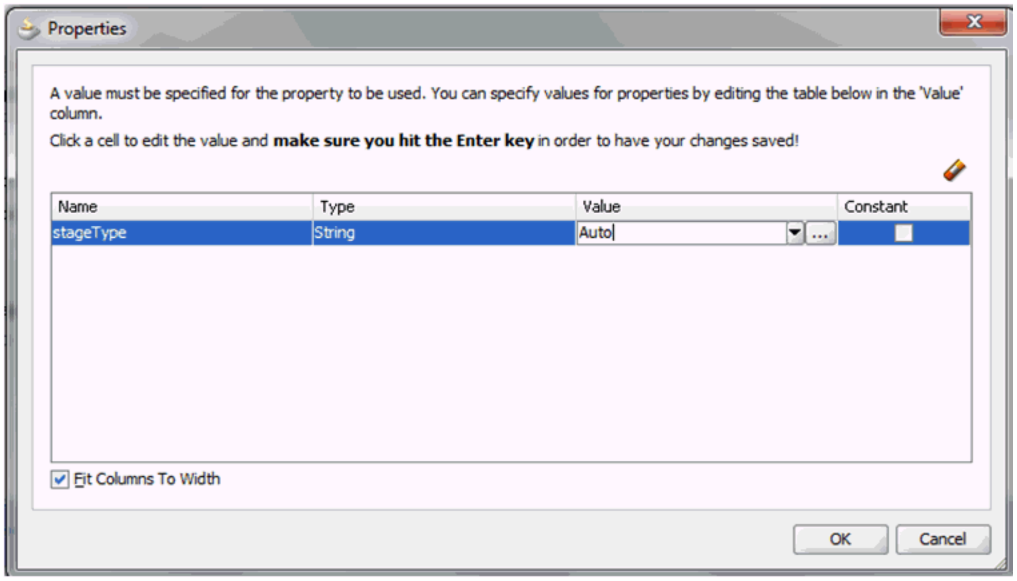


Copy From XPath: /ns17:CatalogData To XPath: /ns17:CatalogData

From	To
catalogDetails_OutputVariable/CatalogData//ns17:CatalogData	catalogData//ns17:CatalogData

16. Switch to the SOA Composite view.
17. Right-click the Business Rule component, and select **Edit**.
18. Click **Create Rule**.
19. Rename the rule from Rule1 to `Auto Approval`.
20. Edit the rule, and specify the stageType property in the Properties dialog box, as shown in Figure 13–17.

Figure 13–17 The stageType Property



A value must be specified for the property to be used. You can specify values for properties by editing the table below in the 'Value' column.

Click a cell to edit the value and **make sure you hit the Enter key** in order to have your changes saved!

Name	Type	Value	Constant
stageType	String	Auto	☐

☑ Fit Columns To Width

To specify the stageType property:

a. Click the **Insert** action below THEN.

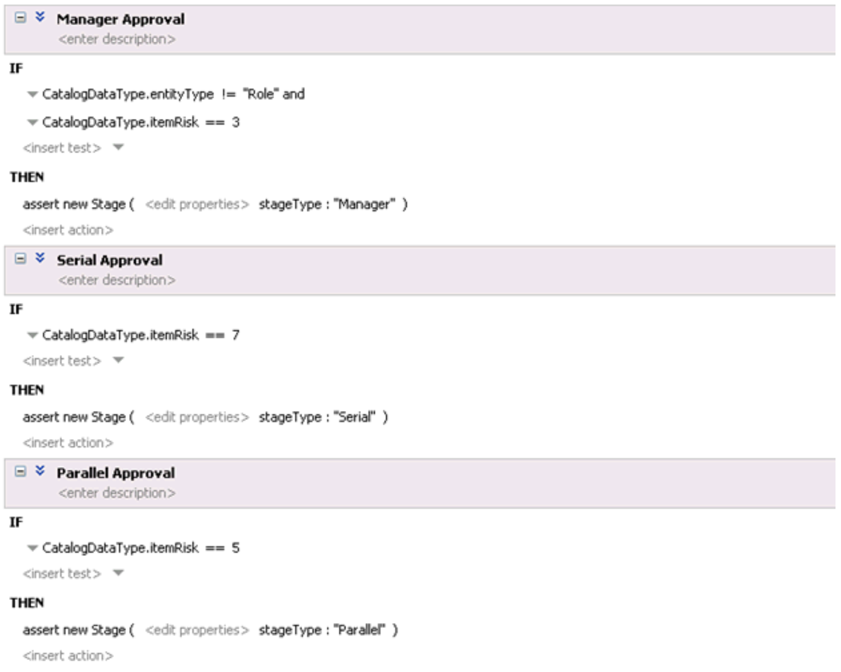
b. Select assert new.

c. Click **<target>** that is added next to assert new, and select **Stage**.

d. Click **<edit properties>**. The Properties dialog box is displayed, as shown in Figure 13–17.

21. Similarly, create the Manager, Serial, and Parallel approval rules, as shown in Figure 13–18.

Figure 13–18 Approval Rules

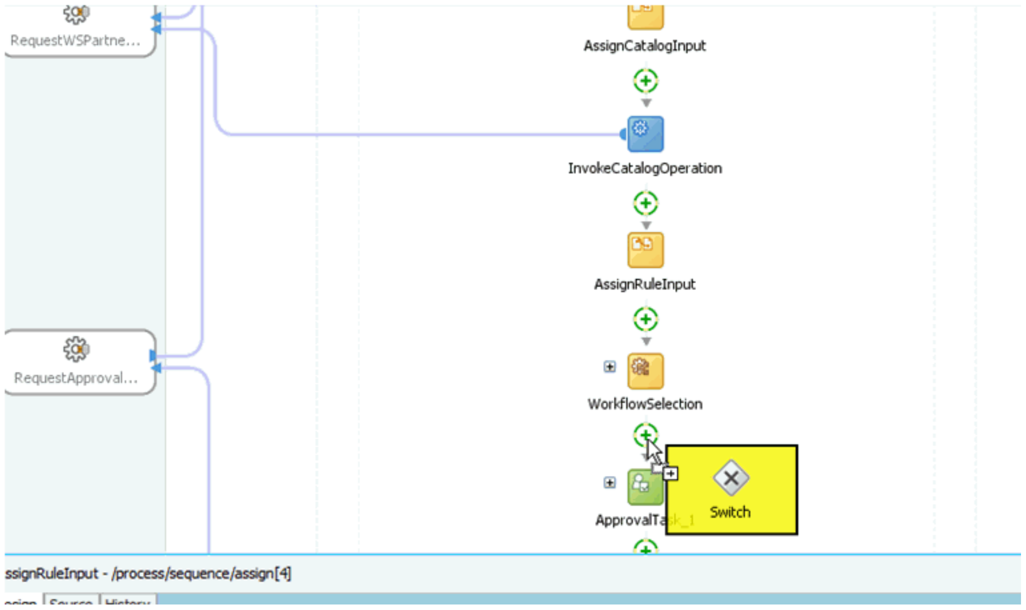


```
Manager Approval
  <enter description>
IF
  CatalogDataType.entityType != "Role" and
  CatalogDataType.itemRisk == 3
  <insert test>
THEN
  assert new Stage ( <edit properties> stageType : "Manager" )
  <insert action>
Serial Approval
  <enter description>
IF
  CatalogDataType.itemRisk == 7
  <insert test>
THEN
  assert new Stage ( <edit properties> stageType : "Serial" )
  <insert action>
Parallel Approval
  <enter description>
IF
  CatalogDataType.itemRisk == 5
  <insert test>
THEN
  assert new Stage ( <edit properties> stageType : "Parallel" )
  <insert action>
```

22. Switch to the BPEL process.

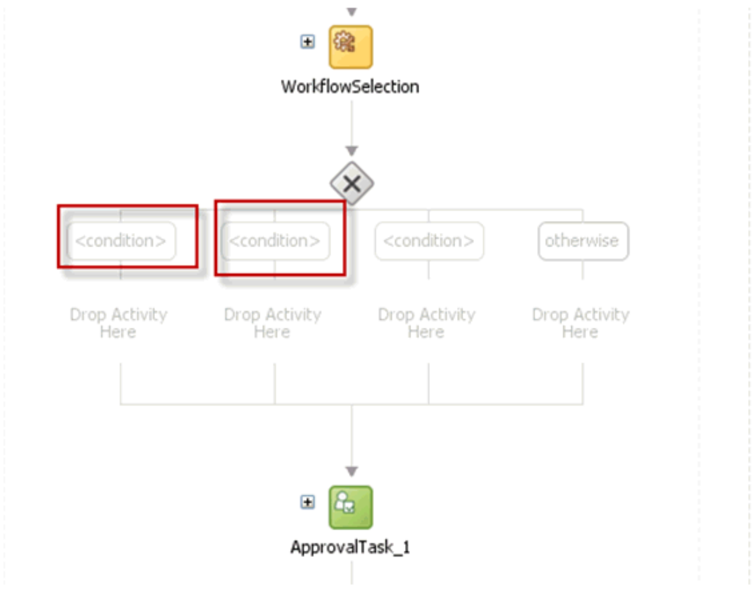
23. Add a switch activity after the WorkflowSelection rule, as shown in Figure 13–19.

Figure 13–19   Switch Activity



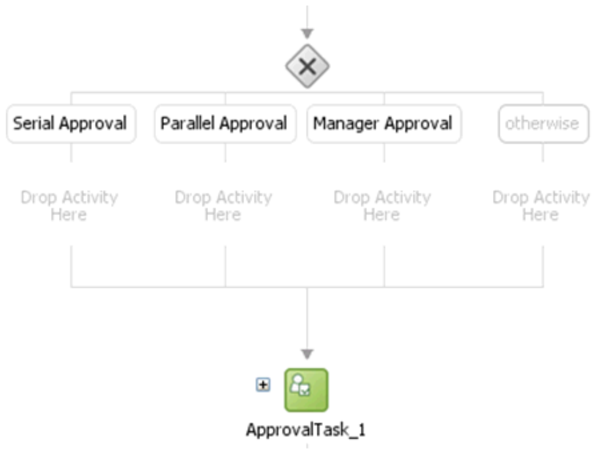
24. Select the Switch activity and add two Switch Case steps, as shown in Figure 13–20.

Figure 13–20   Switch Case Steps



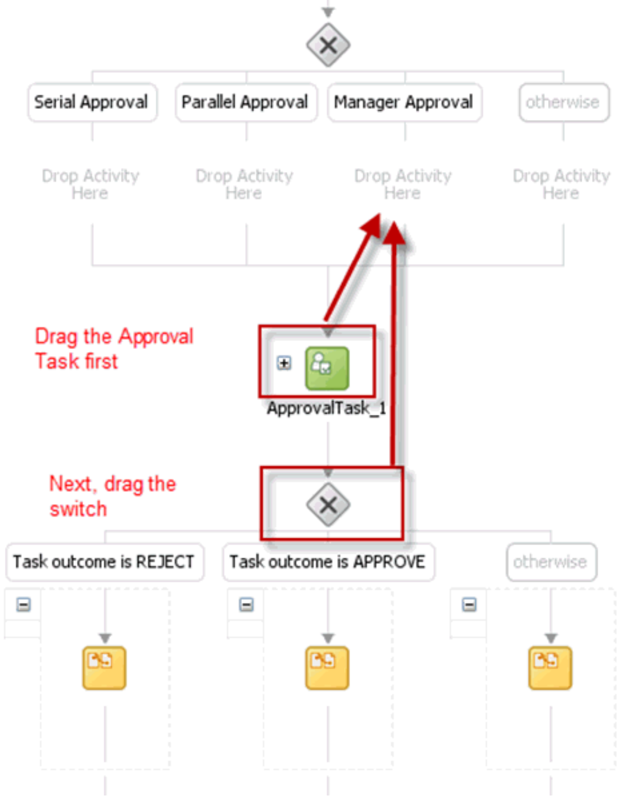
25. Rename the conditions as Serial Approval, Parallel Approval, and Manager Approval, as shown in Figure 13–21.

Figure 13–21 Renamed Conditions



26. Drag the default Human Task into the Manager Switch Case, as shown in Figure 13–22.

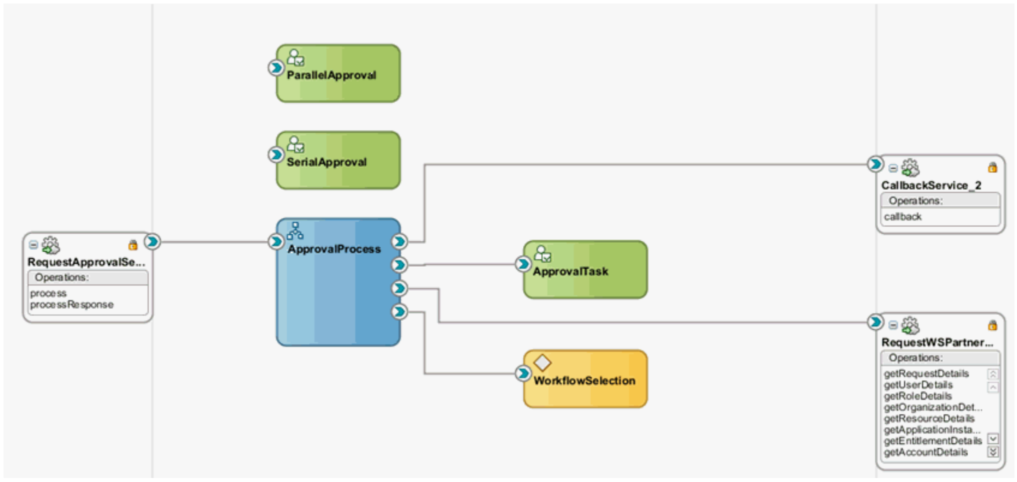
Figure 13–22 Dragging Default Human Task



27. Switch to the SOA Composite view.

28. Add two Human Tasks, SerialApproval and ParallelApproval, as shown in Figure 13–23.

Figure 13–23 Adding Human Tasks



29. Switch to the BPEL Process.
30. Edit the Manager Approval Switch case, and add the following expression:

```
bpws:getVariableData('workflowtype','/ns18:StageOutput/ns18:stageType')='Manage
r'
```

You must first configure the newly added Tasks and then wire them to the BPEL Process.

Note: Oracle recommends using expression builder to add the expression..

13.4.5.4 Configuring Human Tasks

Configuring the Human Task consists of the following:

- Configuring the Parallel Human Task
- Configuring the Serial Approval Task
- Configuring the Default Approval Task

13.4.5.4.1 Configuring the Parallel Human Task

To configure the parallel Human Task:

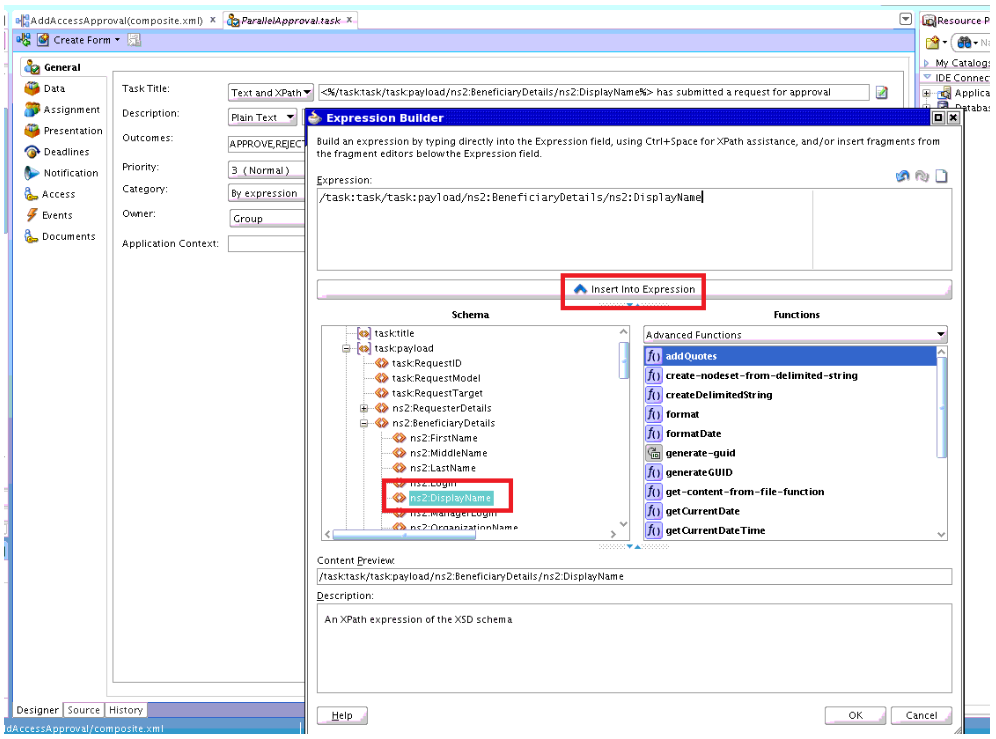
1. Switch to the SOA Composite view.
2. Edit the Parallel Approval Task.
3. Click the **Data** tab, and add the attributes listed in the following table:

Patameter	Data Type
RequestID	{http://www.w3.org/2001/XMLSchema}string
RequestModel	{http://www.w3.org/2001/XMLSchema}string
RequestTarget	{http://www.w3.org/2001/XMLSchema}string
RequesterDetails	{http://xmlns.oracle.com/request/RequestDetails}RequesterDetails

Patameter	Data Type
BeneficiaryDetails	{http://xmlns.oracle.com/request/RequestDetails}BeneficiaryDetails
ObjectDetails	{http://xmlns.oracle.com/request/RequestDetails}ObjectDetails
OtherDetails	{http://xmlns.oracle.com/request/RequestDetails}OtherDetails
url	{http://xmlns.oracle.com/request/RequestDetails}url
Catalogdata	{http://xmlns.oracle.com/RequestServiceApp/RequestDataService/CatalogData}CatalogData
RequesterDisplayName	{http://www.w3.org/2001/XMLSchema}string
BeneficiaryDisplayName	{http://www.w3.org/2001/XMLSchema}string
Requester	{http://www.w3.org/2001/XMLSchema}string

4. Verify the task parameters in the Data tab.
5. Click the **General** tab.
6. Set the Task Title to `<%/task:task/task:payload/ns2:BeneficiaryDetails/ns2:DisplayName%> has submitted a request for approval`. To do so:
   a. Click **Edit** next to Task Title, and select **task:payload**, **ns2:BeneficiaryDetails**, **ns2:DisplayName**.
   b. Click **Insert Into Expression**. Task Title is displayed as shown in Figure 13–24:

Figure 13–24 The Task Title



Task Title: Text and XPath `<%/task:task/task:payload/ns2:BeneficiaryDetails/ns2:DisplayName%> has submitted a request for approval`

Description: Plain Text

Outcomes: APPROVE,REJECT

Priority: 3 (Normal)

Category: By expression

Owner: Group

Application Context:

Build an expression by typing directly into the Expression field, using Ctrl+Space for XPath assistance, and/or insert fragments from the fragment editors below the Expression field.

Expression:

```
/task:task/task:payload/ns2:BeneficiaryDetails/ns2:DisplayName
```

Schema:

```
task:title
task:payload
    task:RequestID
    task:RequestModel
    task:RequestTarget
    ns2:RequesterDetails
    ns2:BeneficiaryDetails
        ns2:FirstName
        ns2:MiddleName
        ns2:LastName
        ns2:Login
        ns2:DisplayName
        ns2:ManagerLogin
        ns2:OrganizationName
```

Functions: Advanced Functions

addQuotes
create-nodeset-from-delimited-string
createDelimitedString
format
formatDate
generate-guid
generateGUID
get-content-from-file-function
getCurrentDate
getCurrentDateTime

Content Preview: /task:task/task:payload/ns2:BeneficiaryDetails/ns2:DisplayName

Description: An XPath expression of the XSD schema

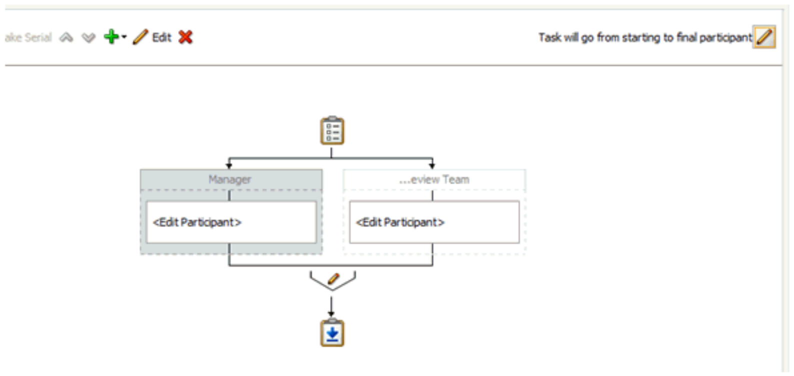
```
<%/task:task/task:payload/ns2:BeneficiaryDetails/ns2:DisplayName%>
```

This can be edited to configure meaningful title, such as:

```
<%/task:task/task:payload/ns2:BeneficiaryDetails/ns2:DisplayName%> has
submitted a request for approval.
```

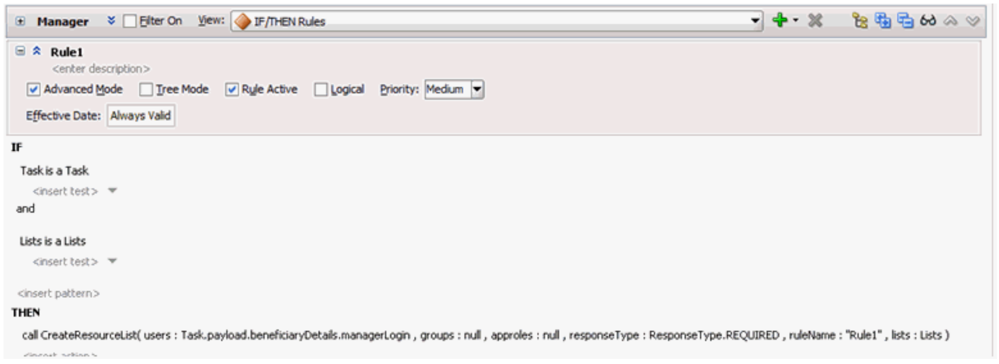
7. Set the Task Owner to Group and SYSTEM ADMINISTRATORS.
8. Click the **Notification** tab, and then click **Advanced**.
9. Select the **Make notification actionable** option.
10. Click the **Assignment** tab.
11. Add a Parallel stage. To do so, select Stage1 in the flow chart, and click the plus (+) icon, and select **Parallel stage**.
12. Select a stage and click **Edit**. Provide the name as `Manager`.
13. Select the other stage, and provide the name as `Review Team`, as shown in Figure 13–25.

Figure 13–25 Manager and Review Team Stages



14. Click the pencil icon after the two stages.
15. In the Vote Outcome dialog box, set the voted outcome to APPROVE. Set the default outcome to REJECT. This is because when both the approver reject the task, the request moves to completed state based on the default outcome.
16. Select the **Share attachments and comments** option.
17. In the Vote Outcome dialog box, click **OK**.
18. Ensure that the **Immediately trigger voted outcome when minimum percentage is met** option is selected.
19. Select **<edit participant>** in Manager Stage, and click **Edit**.
20. In the Add Participant Type dialog box, set the Participant type to Single. From the Build a list of participants using list, select **Rule-based** to build the participant list using Rules.
21. In List RuleSet field, enter Manager. Also, edit the label as Manager.
22. In the Add Participant dialog box, click **OK**. The ParallelApprovalRules.rules tab is displayed.
23. Create a rule, as shown in Figure 13–26.

Figure 13–26 Manager Participant Rule



Manager | [ ] Filter On | View: IF/THEN Rules

Rule1
<enter description>
[✓] Advanced Mode [ ] Tree Mode [✓] Rule Active [ ] Logical Priority: Medium
Effective Date: Always Valid

IF
Task is a Task
<insert test>
and
Lists is a Lists
<insert test>
<insert pattern>
THEN

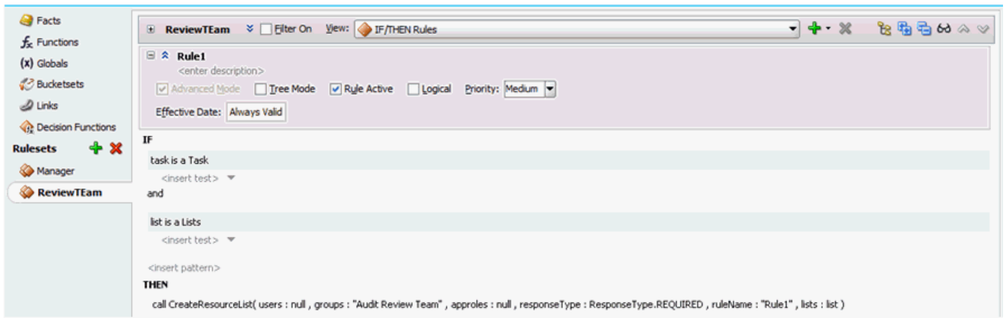
```
call CreateResourceList( users : Task.payload.beneficiaryDetails.managerLogin , groups : null , approles : null , responseType : ResponseType.REQUIRED , ruleName : "Rule1" , lists : Lists )
```

Note: In the Create Rule page, the <insert pattern > in the IF clause is not displayed by default. To display <insert pattern>, expand Rule1 by clicking the two down arrows before Rule1 label, and select the Advanced Mode option as shown in Figure 13–26.

---

24. Similarly, configure the Review Team stage by specifying the following values in the Add Participant Type dialog box:
    - Type: **Single**
    - Label: **Review Team**
    - Build a list of participants using: **Rule-based**
    - List Ruleset: **ReviewTeam**
    - Select **Let participants manually claim the task**
25. Create a participant rule, as shown in Figure 13–27.

Figure 13–27 Review Team Participant Rule



ReviewTEam | [ ] Filter On | View: IF/THEN Rules

Rule1
<enter description>
[✓] Advanced Mode [ ] Tree Mode [✓] Rule Active [ ] Logical Priority: Medium
Effective Date: Always Valid

IF
task is a Task
<insert test>
and
list is a Lists
<insert test>
<insert pattern>
THEN

```
call CreateResourceList( users : null , groups : "Audit Review Team" , approles : null , responseType : ResponseType.REQUIRED , ruleName : "Rule1" , lists : list )
```

13.4.5.4.2 Configuring the Serial Approval Task

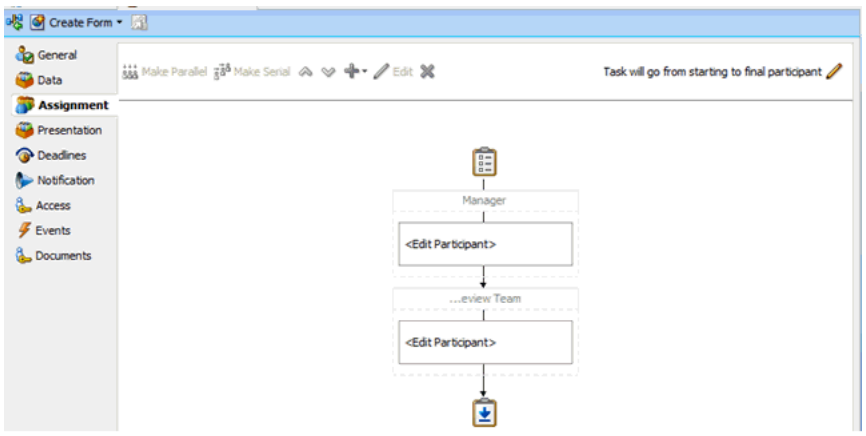
To configure the serial approval task:

1. Switch to the SOA Composite view.
2. Edit the Serial Approval Task.
3. Click the **Data** tab.
4. Add the parameters listed in the following table:

Parameter	Data Type
RequestID	{http://www.w3.org/2001/XMLSchema}string
RequestModel	{http://www.w3.org/2001/XMLSchema}string
RequestTarget	{http://www.w3.org/2001/XMLSchema}string
RequesterDetails	{http://xmlns.oracle.com/request/RequestDetails}RequesterDetails
BeneficiaryDetails	{http://xmlns.oracle.com/request/RequestDetails}BeneficiaryDetails
ObjectDetails	{http://xmlns.oracle.com/request/RequestDetails}ObjectDetails
OtherDetails	{http://xmlns.oracle.com/request/RequestDetails}OtherDetails
url	{http://xmlns.oracle.com/request/RequestDetails}url
Catalogdata	{http://xmlns.oracle.com/RequestServiceApp/RequestDataService/CatalogData}CatalogData
RequesterDisplayName	{http://www.w3.org/2001/XMLSchema}string
BeneficiaryDisplayName	{http://www.w3.org/2001/XMLSchema}string
Requester	{http://www.w3.org/2001/XMLSchema}string

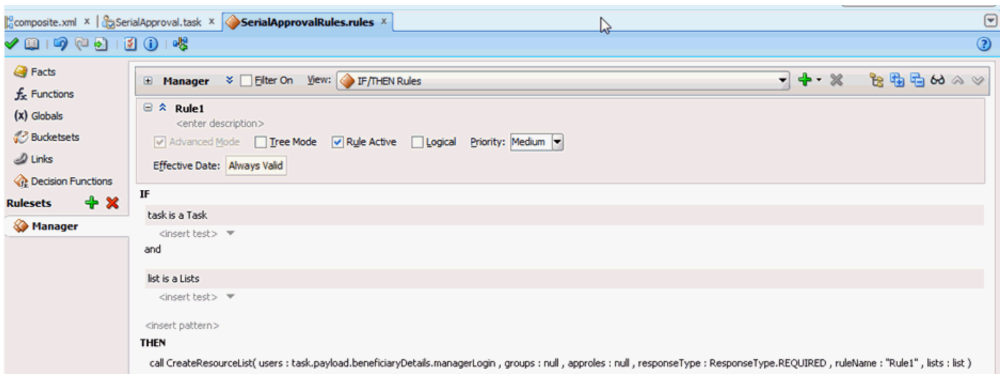
5. Verify the task parameters in the Data tab.
6. Click the **General** tab.
7. Set the Task Title to `<%/task:task/task:payload/ns2:BeneficiaryDetails/ns2:DisplayName%> has submitted a request for approval.`
8. Set the Task Owner to Group and SYSTEM ADMINISTRATORS.
9. Click the **Notification** tab, and then click **Advanced**.
10. Select the **Make notification actionable** option.
11. Click the **Assignment** tab.
12. Add a Sequential stage and rename the stages as Manager and Review Team, as shown in Figure 13–28.

Figure 13–28 Serial Stages



13. Edit the Manager stage.
14. In the Add Participant Type dialog box, set the Participant type to Single and build the participant list using Rules.
15. Create the participant list rule as shown in Figure 13–29.

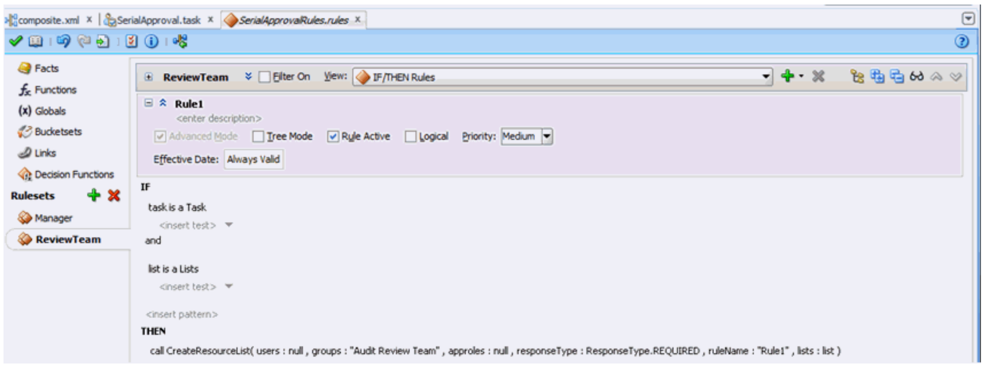
Figure 13–29 Rule for Manager Stage



```
Rule1
<enter description>
Advanced Mode  Tree Mode  [✓] Rule Active  Logical  Priority: Medium
Effective Date: Always Valid
IF
task is a Task
    <insert test>
and
list is a Lists
    <insert test>
<insert pattern>
THEN
call CreateResourceList( users : task.payload.beneficiaryDetails.managerLogin , groups : null , approles : null , responseType : ResponseType.REQUIRED , ruleName : "Rule1" , lists : list )
```

16. Similarly, configure the Review Team stage.
17. Create the participant list rule as shown in Figure 13–30.

Figure 13–30 Rule for Review Team Stage



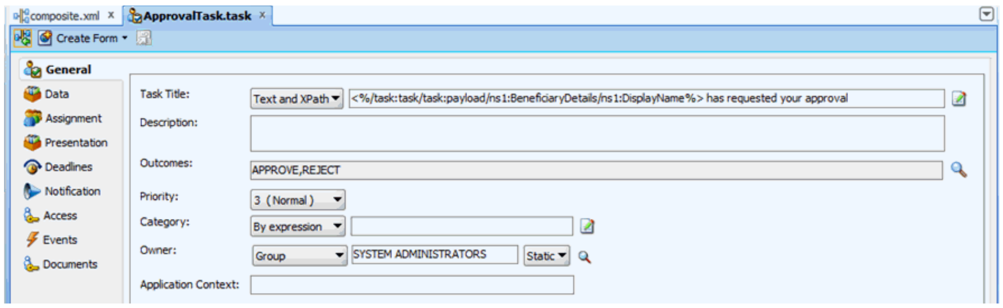
```
Rule1
<enter description>
Advanced Mode  Tree Mode  [✓] Rule Active  Logical  Priority: Medium
Effective Date: Always Valid
IF
task is a Task
    <insert test>
and
list is a Lists
    <insert test>
<insert pattern>
THEN
call CreateResourceList( users : null , groups : "Audit Review Team" , approles : null , responseType : ResponseType.REQUIRED , ruleName : "Rule1" , lists : list )
```

13.4.5.4.3 Configuring the Default Approval Task

To configure the default approval task:

1. Switch to SOA composite view.
2. Edit the Approval Task.
3. Click the **General** tab.
4. Set the task title, as shown in Figure 13–31.

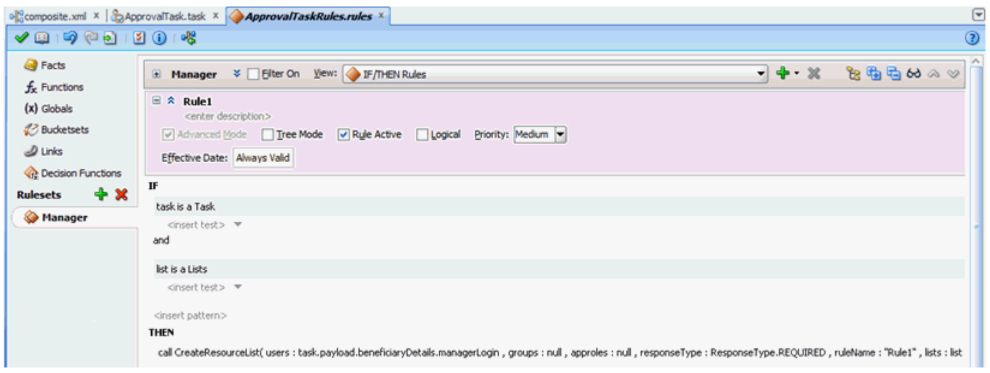
Figure 13–31 Default Approval Task



Task Title: Text and XPath <%/task:task/task:payload/ns1:BeneficiaryDetails/ns1:DisplayName%> has requested your approval
Description:
Outcomes: APPROVE,REJECT
Priority: 3 (Normal)
Category: By expression
Owner: Group SYSTEM ADMINISTRATORS Static
Application Context:

5. Click **Assignment**.
6. Select Stage1.Participant1, and click **Edit**.
7. In the Add Participant Type dialog box, set the Participant type to Single. From the Build a list of participants using list, select **Rule-based** to build the participant list using Rules.
8. Enter Manager in Label and RuleSet, and click **OK**, which opens the rules.
9. Create a Participant list rule, as shown in Figure 13–32.

Figure 13–32 Participant List Rule



Manager Filter On View: IF/THEN Rules
Rule1
<enter description>
☑ Advanced Mode ☐ Tree Mode ☑ Rule Active ☐ Logical Priority: Medium
Effective Date: Always Valid
IF
task is a Task
<insert test>
and
list is a Lists
<insert test>
<insert pattern>
THEN
call CreateResourceList(users : task.payload.beneficiaryDetails.managerLogin , groups : null , approles : null , responseType : ResponseType.REQUIRED , ruleName : "Rule1" , lists : list

13.4.5.5 Configuring the Human Task and BPEL Mappings

Configuring the Human Task and BPEL mappings involves:

- Configuring the Serial Approval Human Task
- Configuring the Parallel Human Task
- Configuring Auto Approval

13.4.5.5.1 Configuring the Serial Approval Human Task

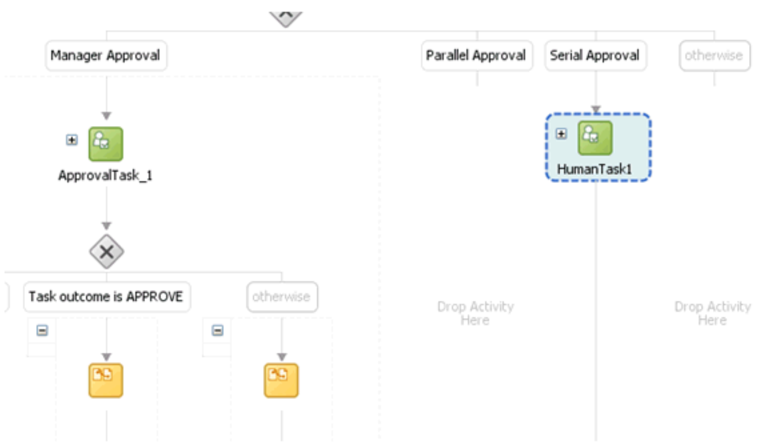
To configure the serial approval Human Task:

1. Switch to BPEL process.
2. Add the following condition to the Serial Approval switch:

```
bpws:getVariableData('workflowtype','/ns18:StageOutput/ns18:stageType') =
'Serial'
```

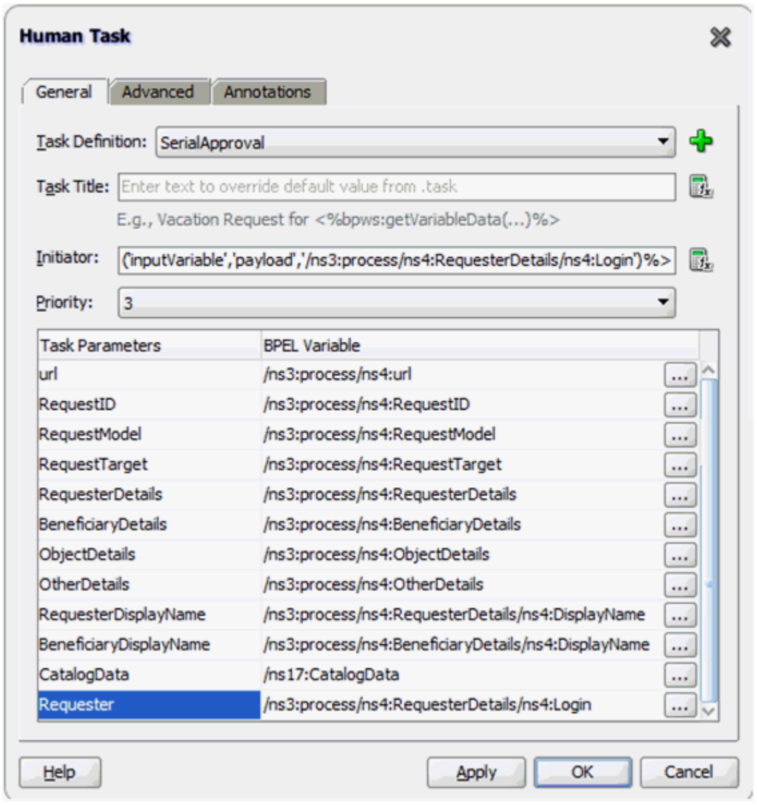
3. Drag and drop a Human Task activity from the SOA Components into the Serial Approval switch, as shown in Figure 13–33.

Figure 13–33 Human Task Activity



4. Edit the Human Task, and in the Human Task dialog box, select the Serial Approval Human Task definition.
5. Map Initiator to requester login, and map the task parameters to the BPEL variable as shown in Figure 13–34.

Figure 13–34 Task Parameters and BPEL Variable Mapping



Task Definition: SerialApproval

Task Title: Enter text to override default value from .task

E.g., Vacation Request for <%bpws:getVariableData(...)%>

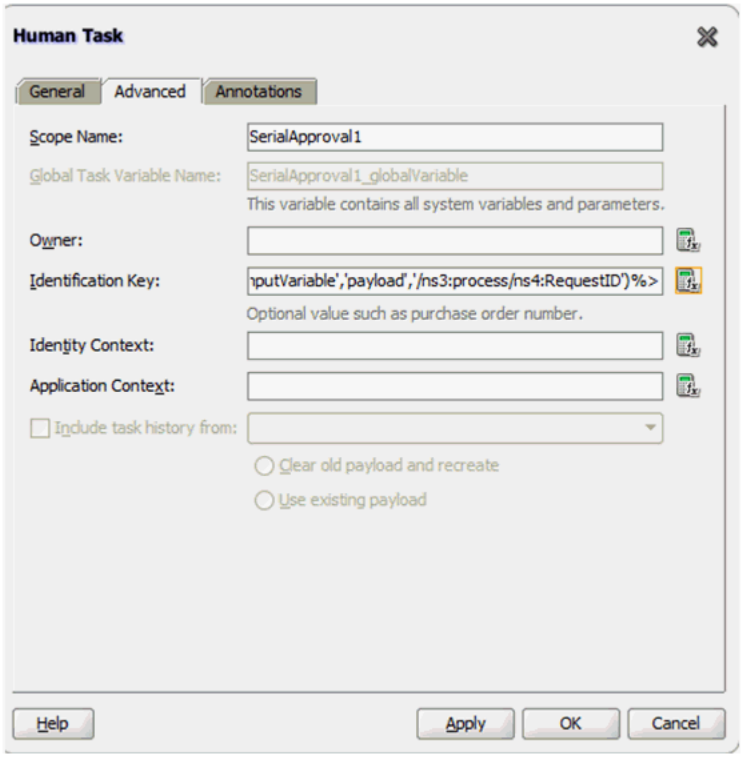
Initiator: ('inputVariable','payload','/ns3:process/ns4:RequesterDetails/ns4:Login')%>

Priority: 3

Task Parameters	BPEL Variable
url	/ns3:process/ns4:url
RequestID	/ns3:process/ns4:RequestID
RequestModel	/ns3:process/ns4:RequestModel
RequestTarget	/ns3:process/ns4:RequestTarget
RequesterDetails	/ns3:process/ns4:RequesterDetails
BeneficiaryDetails	/ns3:process/ns4:BeneficiaryDetails
ObjectDetails	/ns3:process/ns4:ObjectDetails
OtherDetails	/ns3:process/ns4:OtherDetails
RequesterDisplayName	/ns3:process/ns4:RequesterDetails/ns4:DisplayName
BeneficiaryDisplayName	/ns3:process/ns4:BeneficiaryDetails/ns4:DisplayName
CatalogData	/ns17:CatalogData
Requester	/ns3:process/ns4:RequesterDetails/ns4:Login

6. Click the **Advanced** tab.
7. Map the Identification Key to the Request ID as shown in Figure 13–35.

Figure 13–35 Identification Key and Requester ID Mapping



Scope Name: SerialApproval1

Global Task Variable Name: SerialApproval1_globalVariable

This variable contains all system variables and parameters.

Owner:

Identification Key: ıputVariable','payload','/ns3:process/ns4:RequestID')%>

Optional value such as purchase order number.

Identity Context:

Application Context:

Include task history from:

Clear old payload and recreate

Use existing payload

8. Map Initiator to requester login, and then click **Apply** and **OK**.
9. Select the Switch case for Task outcome is REJECT.
10. Replace the existing condition script with the following:

```
bpws:getVariableData('SerialApproval1_globalVariable','payload','/task:task/tas
k:systemAttributes/task:outcome') = 'REJECT'
```

11. Select and edit the Assign activity under Task outcome is REJECT.
12. Delete all the copy rules except one. The copy rule that you retain can be any one so that you can replace it in the Source view.
13. Save and click the **Source** tab.
14. Select the copy activity.
15. Replace the activity with the following:

```
<sequence>
   <assign>
      <copy>
                <from expression="string('rejected')"/>
                <to variable="outputVariable"
                    part="payload"
                    query="/ns3:processResponse/ns3:result"/>
      </copy>
      <copy>
                <from expression="ora:getConversationId()"/>
                <to variable="Invoke_1_callback_InputVariable_1"
                    part="parameters"
                    query="/ns1:callback/arg0"/>
      </copy>
      <copy>
```

```
                <from expression="string('rejected')"/>
                <to variable="Invoke_1_callback_InputVariable_1"
                    part="parameters"
                    query="/ns1:callback/arg1"/>
         </copy>
      </assign>
   </sequence>
```

16. Repeat the steps for the Task outcome is APPROVE. Select the Switch Case and copy the following in the Condition field:

```
bpws:getVariableData('SerialApproval1_globalVariable','payload','/task:task/tas
k:systemAttributes/task:outcome') = 'APPROVE'
```

17. Select the Assign activity under the Approve outcome, and replace the copy rules with the following:

```
<sequence>
          <assign>
            <copy>
                  <from expression="string('approved')"/>
                  <to variable="outputVariable"
                      part="payload"
                      query="/ns3:processResponse/ns3:result"/>
            </copy>
            <copy>
                <from expression="ora:getConversationId()"/>
                <to variable="Invoke_1_callback_InputVariable_1"
                    part="parameters"
                    query="/ns1:callback/arg0"/>
            </copy>
            <copy>
               <from expression="string('approved')"/>
               <to variable="Invoke_1_callback_InputVariable_1"
                   part="parameters"
                   query="/ns1:callback/arg1"/>
            </copy>
          </assign>
</sequence>
```

18. Select the Assign activity under the Otherwise outcome, and replace the copy rules with the following:

```
<sequence>
    <assign>
        <copy>
             <from
expression="bpws:getVariableData('SerialApproval1_globalVariable','payload','/t
ask:task/task:systemAttributes/task:state')"/>
             <to variable="outputVariable" part="payload"
                        query="/ns3:processResponse/ns3:result"/>
         </copy>
         <copy>
            <from expression="ora:getConversationId()"/>
            <to variable="Invoke_1_callback_InputVariable_1"
                part="parameters"
                query="/ns1:callback/arg0"/>
         </copy>
         <copy>
            <from
expression="bpws:getVariableData('SerialApproval1_globalVariable','payload','/t
ask:task/task:systemAttributes/task:state')"/>
            <to variable="Invoke_1_callback_InputVariable_1"
                part="parameters"
                query="/ns1:callback/arg1"/>
         </copy>
         </assign>
</sequence>
```

13.4.5.5.2 Configuring the Parallel Human Task

To configure the parallel Human Task:

1. Add the following condition to the Parallel Approval switch activity:

   ```
   bpws:getVariableData('workflowtype','/ns18:StageOutput/ns18:stageType') =
   'Parallel'
   ```

2. Drag and drop a Human Task activity from the SOA Components into the Parallel Approval switch.
3. Select the Parallel Approval Human Task.
4. Map the Human Task parameters in the same way as the Serial Human Task.
5. Map the Assign activity for the APPROVE outcome in the same way as the equivalent in the Serial Human Task.
6. Map the Assign activity for the REJECT outcome in the same way as the equivalent in the Serial Human Task.
7. Map the Assign activity for the Otherwise outcome in the same way as the equivalent in the Serial Human Task.

Note: You must specify appropriate global variable (ParallelApproval1_globalVariable) in the copy activity.

13.4.5.5.3 Configuring Auto Approval

To configure auto approval:

1. Drag and drop an Assign activity in the Otherwise switch case.
2. Select the Assign activity, and switch to Source view.
3. In the Assign activity, replace the following:

   ```
   <assign name="Assign1"/>
   ```

   With:

   ```
   <sequence>
             <assign>
               <copy>
                     <from expression="string('approved')"/>
                     <to variable="outputVariable"
                         part="payload"
                         query="/ns3:processResponse/ns3:result"/>
               </copy>
               <copy>
                   <from expression="ora:getConversationId()"/>
                   <to variable="Invoke_1_callback_InputVariable_1"
                       part="parameters"
   ```

```
                    query="/ns1:callback/arg0"/>
            </copy>
            <copy>
               <from expression="string('approved')"/>
               <to variable="Invoke_1_callback_InputVariable_1"
                   part="parameters"
                   query="/ns1:callback/arg1"/>
          </copy>
       </assign>
</sequence>
```

13.4.5.6 Deploying the SOA Composite

To deploy the SOA composite:

1. Select **File**, **Save All** to save your work.
2. Right-click the project, and select **Deploy**, ***COMPOSITE_NAME***, **Deploy to Application server**. Alternatively, you can deploy to SAR (SOA Archive), and then deploy it by using Oracle Enterprise Manager.

> **Note:**
>
> - The default version is 1.0. You can also change the version, if you have existing composite instances running.
> - If you are redeploying the composite and you have added or removed one or more human tasks, then it is recommended to deploy with a different version.

13.4.5.7 Creating the Workflow Rules

The SOA composite that you have created can be used for the single and bulk operations. To ensure that the composite is invoked for particular operations, you must create a workflow rule in Oracle Identity Manager.

To create workflow rules in Oracle Identity Manager:

1. Login to Oracle Identity System Administration.
2. On the left navigation pane, under Workflows, click **Approval**.
3. Create a workflow rule for the Bulk Provision Application Instance operation, and set it to auto approve. See "Configuring Approval Workflow Rules" in *Administering Oracle Identity Manager* for information about creating and managing workflow rules.
4. Create a workflow rule for the Provision Application Instance operation, and specify that the composite you deployed will be invoked.

> **Note:** While it is possible to create multiple SOA composites for each type of request, it is recommended that you use a single SOA composite (as demonstrated in this tutorial) and create multiple Human Tasks. You can use rules created by using Oracle Business Rules to pick a Human Task (as demonstrated in this tutorial).

Tip: You can access custom attribute's value of entities, such as catalog, user, role, or organization, supported by the request web service is SOA composite BPEL process. The custom attributes or UDFs are part of the CustomAttribute element. An instance of catalog entity containing UDF is:

```
<ns12:CustomAttribute Name="ApproverRolePhoneNumber">
    <ns5:Value>1234</ns5:Value>
</ns12:CustomAttribute>
<ns12:CustomAttribute Name="ApproverRoleEmailId">
   <ns5:Value>approver@mydomain.com</ns5:Value>
</ns12:CustomAttribute>
```

For example, to access the ApproverRolePhoneNumber catalog UDF value in BPEL process, specify the following:

```
bpws:getVariableData('catalogDetails','CatalogData','/ns22:CatalogD
ata/ns22:CustomAttribute[@Name =
string("ApproverRolePhoneNumber")]/ns24:Value')
```

13.5 Configuring Default Approval Composites for Single and Bulk Operations

The request-level composite is applicable to bulk requests, and the operation-level composite is applicable to single and child requests.

You can configure the default composites by setting the DefaultRequestLevelComposite and DefaultOperationLevelComposite properties in the oim-config.xml file. You can edit these properties by using System MBean Browser in Oracle Enterprise Manager. The default values for these properties are `default/DefaultRequestApproval!3.0` and `default/DefaultOperationalApproval!3.0` respectively.

The values for these properties are in the following format:

```
NAMESPACE/COMPOSITE_NAME!VERSION
```

For example:

```
default/AddAccessApproval!2.0
```

If you change the default values of the DefaultRequestLevelComposite and DefaultOperationLevelComposite properties, then you must restart Oracle Identity Manager.

13.6 Creating and Deploying Custom Task Details Taskflow

By default, all tasks are configured to use the default task details page in pending approvals. This taskflow is not customizable. However, you might want to customize the UI or show some other information in the task details page. This section describes how to build your own taskflow, and configure the human task in the DefaultRequestApproval composite to invoke your custom taskflow.

This section contains the following topics:

- Prerequisites for Developing Custom Task Details Taskflow
- Developing Custom Task Details Taskflow

- Developing Custom Task Details for Email Notification (Optional)
- Deploying the Task Details Taskflow
- Configuring Human Task and Taskflow Permissions
- Testing the Custom Taskflow

13.6.1 Prerequisites for Developing Custom Task Details Taskflow

Before developing a custom task details taskflow, you must have the following software installed on your computer:

- Oracle Identity Manager 11*g* Release 2 (11.1.2.3.0)
- Oracle SOA 11*g* (11.1.1.9.0)
- JDeveloper 11*g* (11.1.1.7.0) with Oracle SOA Composite Editor extension

13.6.2 Developing Custom Task Details Taskflow

To build a custom taskflow for the human task in the DefaultRequestApproval composite:

1. Open Jdeveloper and create a new Generic Application. To do so:

   a. Enter Application Name as `RequestApprovalTaskDetailsApp`, and then click **Next**.

   b. Enter Project Name as `RequestApprovalTaskDetails`. Do not select any project technologies.

   c. Click **Finish**.

2. Add Oracle Identity Manager shared library. To do so:

   a. Right-click **RequestApprovalTaskDetails** project, and select **Project Properties**, **Libraries and Classpath**.

   b. Click **Add Library**.

   c. Click **Load Dir**.

   d. Navigate to the *IAM_HOME*/server/jdev.lib/ directory, and click **Select**.

   Note: *IAM_HOME* is the path to the Oracle Identity Manager home directory, for example, BEA_HOME/Oracle_IDM1/. Here, *BEA_HOME* is the path to the middleware directory in Oracle Identity Manager installation.

   ---

   e. Select **OIM View Shared library**, **OIM Model Shared library**, and then click **OK**.

   f. Click **OK**.

3. Create task details taskflow. To do so:

   a. Navigate to the following directory in shiphome:

      IAM_HOME/server/workflows/composites/

   b. Unzip the DefaultRequestApproval.zip file.

   c. Go back to Jdeveloper, right-click **RequestApprovalTaskDetails**, and select **New**.

   d. Select **Web Tier**, **JSF**, ADF task flow based on human task.
   e. In the file browser, navigate to the directory in which you unzipped DefaultRequestApproval.zip. Select the DefaultRequestApproval/ApprovalTask.task file
   f. In the Create Task flow dialog box, provide the following values:

      File Name: request-approval-details-tf.xml

      Task Flow ID: request-approval-details-tf
   g. Click **OK**.

4. Delete hwtaskflow.xml. To do so, go to Application Sources under RequestApprovalTaskDetails project, and then delete hwtaskflow.xml.
5. Create the task details page. To do so:
   a. Open request-approval-details-tf.xml. Switch to diagram mode.
   b. Rename **taskdetails1_jspx** view activity to **request-approval-details**.
   c. Right-click the **request-approval-details** view activity, and select **Create Page**. Provide the following values:

      File name: request-approval-details.jspx

      Initial Page layout and content: Blank Page
   d. Click **OK**.
6. Add managed bean for this page. To do so:
   a. Right-click the **RequestApprovalTaskDetails** project, and select **New**, **Java Class**. Provide the following values:

      Name: RequestApprovalDetailsStateBean

      Package: oracle.iam.ui.custom.view.backing
   b. Click **OK**.
   c. Add the following code to the managed bean:

```
package oracle.iam.ui.custom.view.backing;

import javax.el.ELContext;
import javax.el.ExpressionFactory;
import javax.el.ValueExpression;

import javax.faces.application.Application;
import javax.faces.context.FacesContext;

import oracle.iam.ui.platform.model.config.ConstantsDefinition;


public class RequestApprovalDetailsStateBean implements
java.io.Serializable{
    public RequestApprovalDetailsStateBean() {
        super();
    }

    private String requestAction =
ConstantsDefinition.REQUEST_ACTION_APPROVAL_UPDATE;
    private String requestType =
ConstantsDefinition.REQUEST_TYPE_VIEW_DETAIL;
```

```
    public void setRequestAction(String requestAction) {
        this.requestAction = requestAction;
    }

    public String getRequestAction() {
        return requestAction;
    }

    public void setRequestType(String requestType) {
        this.requestType = requestType;
    }

    public String getRequestType() {
        return requestType;
    }

    public String getUserIds() {
        Object benefDisplayName =
getValueFromELExpression("#{bindings.DisplayName.inputValue}");
        //benefDisplayName would be "None" if beneficiary does not exist
        if (benefDisplayName != null &&
!ConstantsDefinition.NONE_BENEF_DISPLAY_NAME.equalsIgnoreCase(benefDisplayN
ame.toString()))
            return benefDisplayName.toString();

        Object requestTarget =
getValueFromELExpression("#{bindings.RequestTarget.inputValue}");
        if (requestTarget != null)
            return requestTarget.toString();

        return "";
    }

    private Object getValueFromELExpression(String expression) {
        FacesContext facesContext = FacesContext.getCurrentInstance();
        Application app = facesContext.getApplication();
        ExpressionFactory elFactory = app.getExpressionFactory();
        ELContext elContext = facesContext.getELContext();
        ValueExpression valueExp =
            elFactory.createValueExpression(elContext, expression,
                                            Object.class);
        return valueExp.getValue(elContext);
    }


}
```

d. Open request-approval-details-tf.xml in Overview mode. Select Managed Beans sections and register the managed bean with the following details:

Name: requestApprovalDetailsStateBean

Class: oracle.iam.ui.custom.view.backing.RequestApprovalDetailsStateBean

Scope: pageFlow

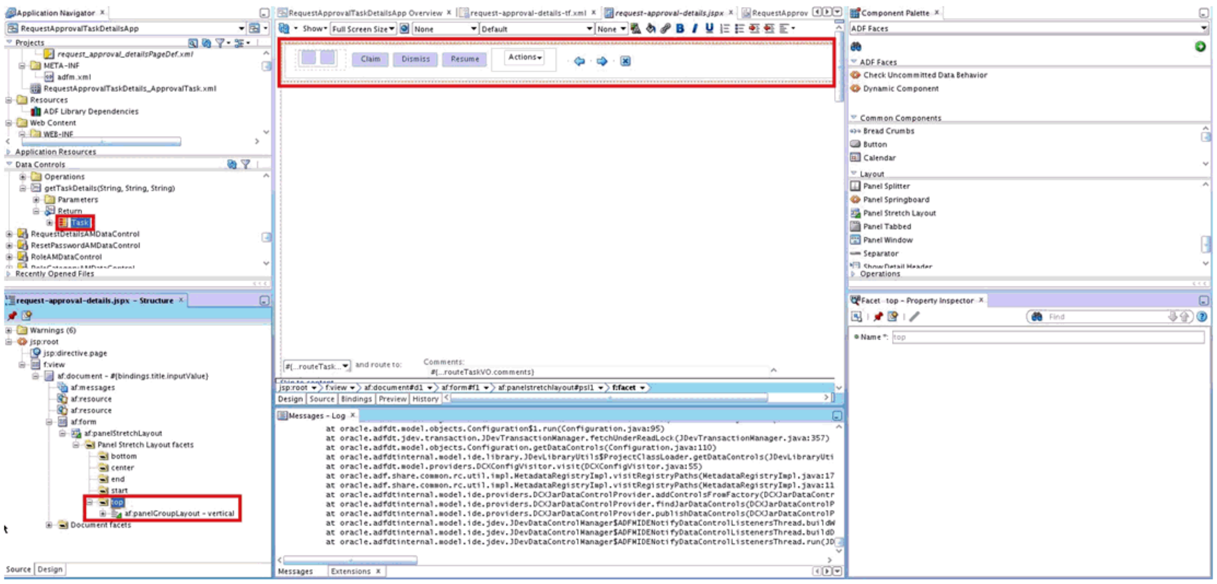
7. Create the details page structure. To do so:

a. Open request-approval-details.jspx.

b. From the Component Palette, add a **panelStretchLayout** to the page. In the Property Inspector, set `TopHeght==auto` for panelStretchLayout.

c. Go to Data controls in Application Navigator. Expand **RequestApprovalTaskDetails_ApprovalTask**, **getTaskDetails**, **Return**. Drag and drop **Task** from Data Controls on to the Top Facet of panelStretchLayout, as shown in Figure 13–36. From the context menu, select **Human Task**, **Task Action**. The Human task actions are added to the Top Facet.

Figure 13–36 Dragging Task to the Top Facet


Application Navigator
RequestApprovalTaskDetailsApp
Data Controls
getTaskDetails(String, String, String)
Parameters
Return
Task
Claim
Dismiss
Resume
Actions
Component Palette
ADF Faces
Panel Stretch Layout
Panel Tabbed
request-approval-details.jspx - Structure
af:panelStretchLayout
Panel Stretch Layout facets
bottom
center
end
start
top
af:panelGroupLayout - vertical
Facet - top - Property Inspector
Name: top

```
at oracle.adfdt.model.objects.Configuration$1.run(Configuration.java:95)
at oracle.adfdt.jdev.transaction.JDevTransactionManager.fetchUnderReadLock(JDevTransactionManager.java:357)
at oracle.adfdt.model.objects.Configuration.getDataControls(Configuration.java:110)
at oracle.adfdtinternal.model.ide.library.JDevLibraryUtils$ProjectClassLoader.getDataControls(JDevLibraryUti
at oracle.adfdt.model.providers.DCXConfigVisitor.visit(DCXConfigVisitor.java:55)
at oracle.adf.share.common.rc.util.impl.MetadataRegistryImpl.visitRegistryPaths(MetadataRegistryImpl.java:17
at oracle.adf.share.common.rc.util.impl.MetadataRegistryImpl.visitRegistryPaths(MetadataRegistryImpl.java:11
at oracle.adfdtinternal.model.ide.providers.DCXJarDataControlProvider.addControlsFromFactory(DCXJarDataContr
at oracle.adfdtinternal.model.ide.providers.DCXJarDataControlProvider.findJarDataControls(DCXJarDataControlP
at oracle.adfdtinternal.model.ide.providers.DCXJarDataControlProvider.publishDataControls(DCXJarDataControlP
at oracle.adfdtinternal.model.ide.jdev.JDevDataControlManager$ADFMIDENotifyDataControlListenersThread.buildW
at oracle.adfdtinternal.model.ide.jdev.JDevDataControlManager$ADFMIDENotifyDataControlListenersThread.buildD
at oracle.adfdtinternal.model.ide.jdev.JDevDataControlManager$ADFMIDENotifyDataControlListenersThread.run(JD
```

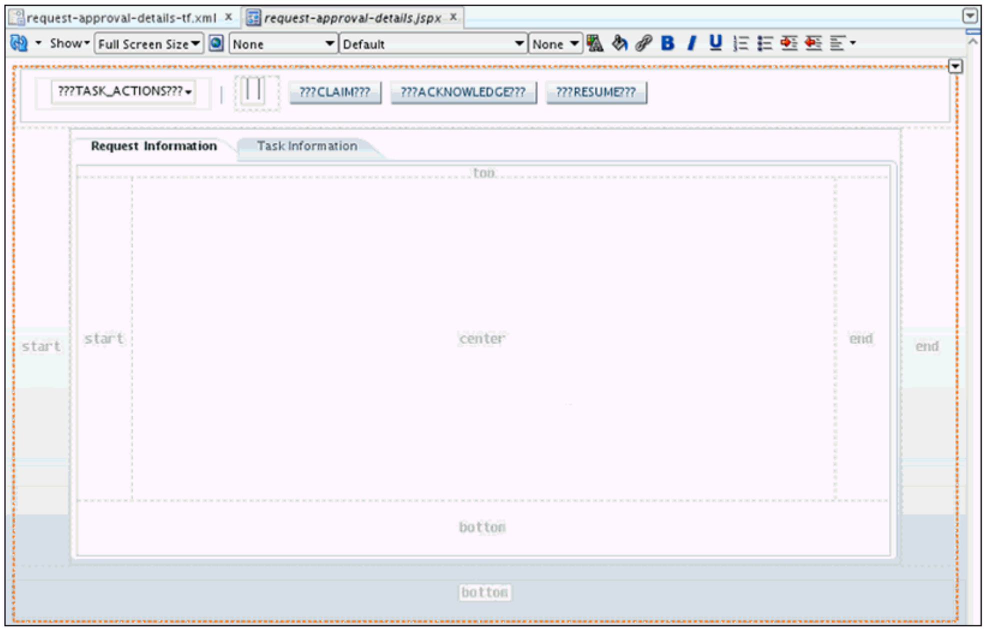
d. From the Component Palette, add a **panelTabbed** layout to the Center Facet of panelStretchLayout.

e. From the Component Palette, add two **showdetailItem** components to the panelTabbed layout. From property inspector, set the text name for these components as `Request Information` and `Task Information`.

f. Click the **Request Information** tab. From the property inspector, set attribute `stretchChildren=first`.

g. Add another **panelStretchLayout** in Request Information tab. Set attribute `topHeight=auto` for this panelStretchLayout. Figure 13–37 shows the Request Information and Task Information tabs.

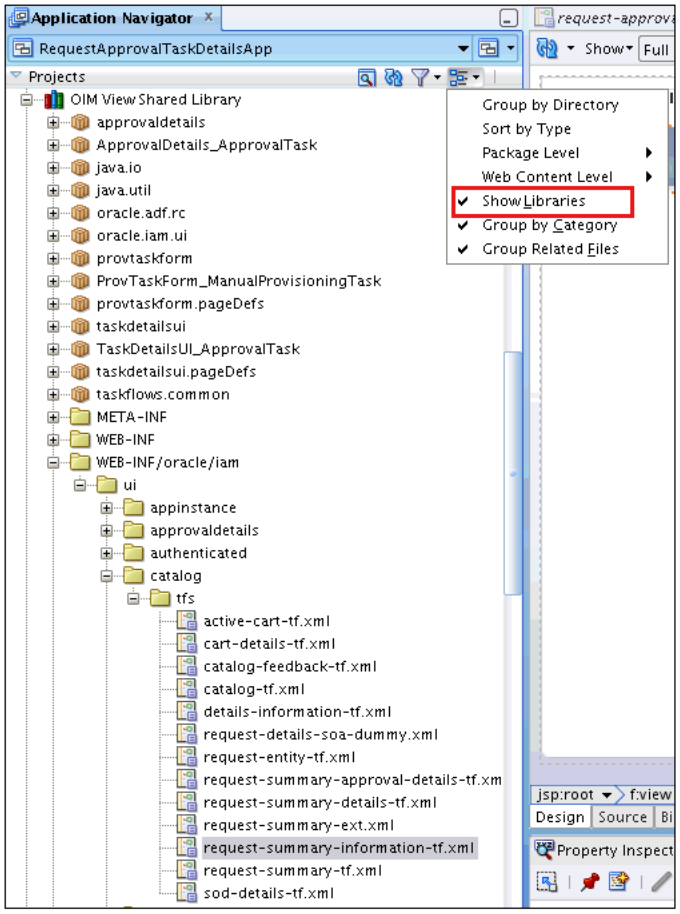
Figure 13–37 The panelTabbed Layout



8. Populate the **Request Information** tab. To do so:

   a. Go to Navigator Display Options, and select **Show Libraries**, shown in Figure 13–38. This will show **OIM View Shared Library** in the Application Navigator.

Figure 13–38 OIM View Shared Library



b. In the Application Navigator, expand **OIM View Shared Library**, **WEB-INF/oracle/iam/ui/catalog/tfs**. Drag and drop **request-summary-information-tf.xml** to the Top Facet of PanelStretchLayout added in step 7g. The Create context menu is displayed. Select **Region**.The Edit Task Flow Binding dialog box is displayed. You can provide parameters to the taskflow later. Therefore, click **OK**.

c. Similarly, drag and drop **catalog-tf.xml** to the Center Facet of PanelStretchLayout added in step 7g. The Create context menu is displayed. Select **Region**. The Edit Task Flow Binding dialog box is displayed. Click **OK**.

d. Click the **Bindings** tab at the bottom of the page to view the bindings. Click the plus (+) sign to add a binding in the following way:

i) Enter the following and click **OK**:

Category: Generic Bindings

Item to be created: attributeValues

ii) Click **Add Datasource**. Select **RequestApprovalTaskDetails_ApprovalTask**, **getTaskDetails**, **Return**, **Task**, **Payload**. Click **OK**.

iii) Specify Attribute as `RequestID`, and click **OK**.

e. Under executables, select **taskflow-requestsummaryinformationtf1**. In the Property Inspector, add a taskflow parameter by clicking the plus (+) sign.

Edit the value field, and update it with #{bindings.RequestID.inputValue}, as shown:

ID=requestID, Value= #{bindings.RequestID.inputValue}

f. Click the plus (+) sign to add another binding. This binding will be referenced in RequestApprovalDetailsStateBean.

i) Enter the following and click **OK**:

Category: Generic Bindings

Item to be created: attributeValues

ii) Click **Add Datasource**. Select **RequestApprovalTaskDetails_ApprovalTask**, **getTaskDetails**, **Return**, **Task**, **Payload**, **BeneficiaryDetails**. Click **OK**.

iii) Specify Attribute as `DisplayName`, and click **OK**.

g. Click the plus (+) sign to add another binding. This binding will be referenced in RequestApprovalDetailsStateBean.

i) Enter the following and click **OK**:

Category: Generic Bindings

Item to be created: attributeValues

ii) From the list, select datasource **RequestApprovalTaskDetails_ApprovalTask**, **getTaskDetails**, **Return**, **Task**, **Payload**.

iii) Specifiy Attribute as `RequestTarget`, and click **OK**.

h. Select **taskflow-catalogtf1** in Executables, click **Edit** on the top-right corner of Executables, and edit the values of the following in the Edit Task flow Binding dialog box:

– Id=requestId, Value= #{bindings.RequestID.inputValue}

– Id=requestType, Value=#{pageFlowScope.requestApprovalDetailsStateBean.requestType}

– Id=requestAction, Value=#{pageFlowScope.requestApprovalDetailsStateBean.requestAction}

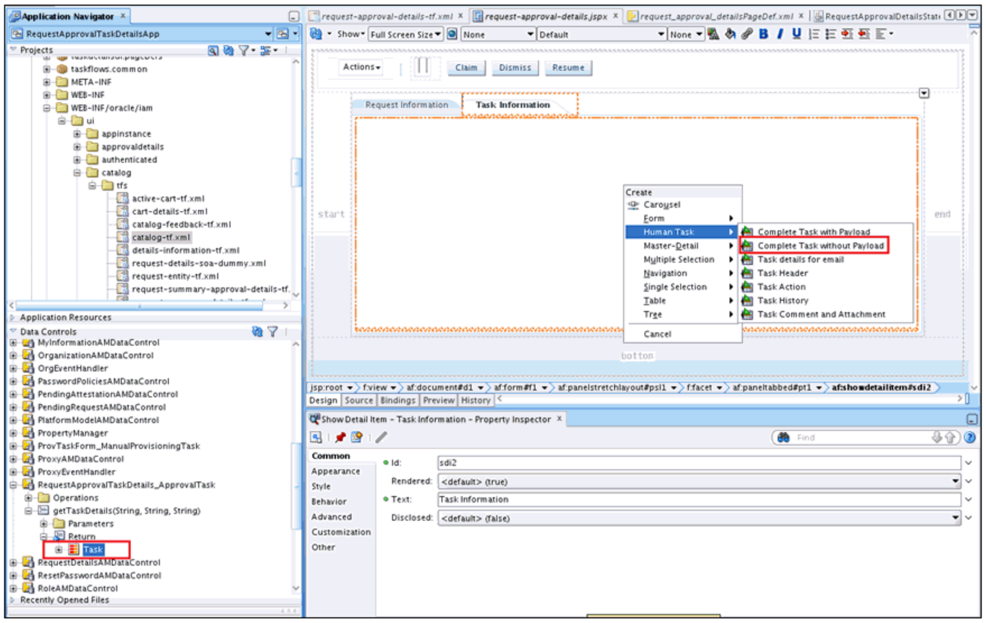
– Id=userIds, Value=#{pageFlowScope.requestApprovalDetailsStateBean.userIds}

9. Populate the **Task Information** tab. To do so:

a. Switch to Design mode. Click the **Task Information** tab.

b. In the Application Navigator, go to Data Controls. Expand **RequestApprovalTaskDetails_ApprovalTask**, **getTaskDetails**, **Return**. Drag and drop **Task** in the **Task Information** tab. The Create context menu is displayed. Select **Human Task**, **Complete Task without Payload**, as shown in Figure 13–39.

Figure 13–39 Task Details DataControl



Id: sdi2

Rendered: <default> (true)

Text: Task Information

Disclosed: <default> (false)

c. A **panelHeader** wrapped inside **panelGroupLayout** is added to the **Task Information** tab. Navigate to the **panelHeader** and delete the **Toolbar Facet** of the **panelHeader**. The task actions have already been in step 7c). In addition, task details, task history, comments, and attachments are also added to the Task Information tab.

d. Save your work.

13.6.3 Developing Custom Task Details for Email Notification (Optional)

By default, for sending email notification, if there is no separate page for email, then the same task details page developed in "Developing Custom Task Details for Email Notification (Optional)" on page 13-46 is sent in email notification.

Sometimes, limited information needs to be sent in email notification. In such scenarios, separate page for email notification can be developed. The email page will also be part of the same task details taskflow.

For more information on building custom taskflow for email, refer to "Creating an Email Notification" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

13.6.4 Deploying the Task Details Taskflow

To deploy the task details taskflow:

1. Deploy the Task Details as an ADF library jar. To do so:

   a. Right-click **RequestApprovalTaskDetails**, **Project Properties**, **Deployment**.

   b. Click **New**. The Create deployment profile dialog box is displayed.

   c. Provide following values, and click **OK**.

   Archive Type: ADF Library Jar File

   Name: adflibRequestApprovalTaskDetails

   d. Right-click **RequestApprovalTaskDetails**, and select **Deploy**, **adflibRequestApprovalTaskDetails**.
   e. In the deployment action popup, click **Finish**.
2. Package the adflibRequestApprovalTaskDetails.jar in custom shared library. To do so:
   a. Navigate to the *IAM_HOME*/server/apps/ directory.
   b. Create following directory structure:

      WEB-INF/lib/
   c. Copy adflibRequestApprovalTaskDetails.jar to the WEB-INF/lib/ directory.
   d. Update *IAM_HOME*/server/apps/oracle.iam.ui.custom-dev-starter-pack.war to add adflibRequestApprovalTaskDetails.jar. For example:

      ```
      jar uvf oracle.iam.ui.custom-dev-starter-pack.war WEB-INF/*
      ```
3. Restart Oracle Identity Manager managed server for the changes to custom shared library to take effect.

13.6.5 Configuring Human Task and Taskflow Permissions

To configure Human Task and taskflow permissions:

1. Add view permission for custom taskflow by using Authorization Policy Manager (APM). To do so:
   a. Login to APM application as WebLogic user.
   b. Navigate to **Applications**, **OracleIdentityManager**, **Resource Types**. Click **Open**.
   c. Click New to create a new resource type. Provide following details:
      – **Display Name:** ADF Taskflows
      – **Name:** ADFTaskFlows
      – **Actions:** personalize, customize, grant, view. Click `New` to add each action.
      – **Supports Resource Hierarchy:** No
      – **Resource Delimiter:** Slash(/)
      – **Evaluation Logic:** Permission Class
      – **Permission Class:** oracle.adf.controller.security.TaskFlowPermission
      – **Action Name Delimiter:** Comma(,)
   d. Click **Save**.
   e. Navigate to **Applications**, **OracleIdentityManager**, **Default Policy Domain**, **Resource Catalog**, **Resources**. Click **Open**.
   f. Click **New** to create a new resource. Provide the following values, and then click **Save**.
      – **Resource Type:** Select the resource type created in step 1(c).
      – **Display Name:** Provide a display name for your custom taskflow.
      – **Name:** Provide the name of the custom taskflow in the following format:

```
TASKFLOW_DOCUMENT#TASKFLOW_ID
```

For example:

```
/WEB-INF/request-approval-details-tf.xml#request-approval-details-tf
```

– **Description:** Provide a description for the custom taskflow.

Note: For each custom taskflow, you must create a resource as mentioned in step 1(f). You can use the same resource type that you created in step 1(c) for all your custom taskflows.

---

g. Navigate to **Applications**, **OracleIdentityManager**, **Default Policy Domain**, **Authorization Policies**. Click **Open**.

h. Select **Find By Principal**, and click **Search**. If you want to add to a displayed existing policy, then select it and click **Open**. Otherwise, click **New** to create a policy.

i. Add **name** and **principals** for the new policy.

j. Click **Add Targets** (+) sign. The Search Targets dialog box is displayed.

k. Click the **Resources** tab. Provide the resource type as defined in step 1(f), and then click Search.

l. Select the resource created in step 1(f). Click **Add Selected**.

m. Click **Add Targets**. The resource is added to the Targets table.

n. Expand the resource that you added to the table. Select the permissions you want to apply to the taskflow.

o. Click **Apply**.

2. Configure human task to use the custom taskflow. To do so:

   a. Login to Oracle Enterprise Manager as WebLogic user.

   b. Navigate to **Farm_*IAM_DOMAIN***, **SOA**, **soa_infra (*SOA_SERVER*)**, **default**, **DefaultRequestApproval [4.0]**.

   c. Click **Component Metrics**, **Approval Task**.

   d. Click the **Administration** tab.

   e. Modify the URI in the existing entry to point to the custom taskflow, as shown:

      – **Application Name:** *ANY_NAME*

      – **Host Name:** *OIM_SERVER_HOSTNAME*

      – **HTTP Port:** *OIM_HTTP_PORT*

      – **HTTPS Port:** *OIM_HTTPS_PORT* (Optional)

      – **URI:** /identity/faces/adf.task-flow?_id=request-approval-details-tf&_document=WEB-INF/request-approval-details-tf.xml

13.6.6 Testing the Custom Taskflow

To test custom taskflow:

1. Login to Oracle Identity Self Service as an end user.
2. Go to My Information, and modify the value of the Telephone attribute. A request is created and the task is assigned to the System Administrator.
3. Login to Oracle Identity Self Service as System Administrator.
4. Go to Pending Approvals.
5. Click the Task Details link of the corresponding request. The custom task details page is displayed.

13.7 Extending Request Management Operations

You can customize certain aspects of request management operations to allow greater flexibility and implement customized logic for additional functionality. To achieve this, you can use request management plug-ins. There are plug-in points that you can use to implement customization.

This section discusses the plug-in points in the following topics:

- Running Custom Code Based on Request Status Change
- Validating Request Data
- Prepopulation of an Attribute Value During Request Creation

13.7.1 Running Custom Code Based on Request Status Change

In Oracle Identity Manager, a request undergoes change in status at each stage of its lifecycle. The request engine exposes a plug-in point that allows running of custom code during request status change. A plug-in with custom code that extends this plug-in point can be implemented and registered for running the code. The plug-in point is the **oracle.iam.request.plugins.StatusChangeEvent** interface with the **public void followUpActions(String reqId)** method. This method consists of the request id parameter, using which the request details can be obtained with the help of request management APIs.

See Also: Chapter 17, "Developing Plug-ins" for detailed information about plug-ins and plug-in points

Any code that is to be run during the status change must be implemented in the followUpActions() method in a plug-in class that implements the oracle.iam.request.plugins.StatusChangeEvent interface. You must specify at which request status change this plug-in is to be run in the plugin.xml file.

For example, when a request in Oracle Identity Manager moves to the Request Failed status, you want to run a custom code that sends a notification to an administrator. To do so:

1. Create a new plug-in class with name RequestFailedChangeEvent that implements the oracle.iam.request.plugins.StatusChangeEvent interface. This class must have the logic of sending a notification to the administrator in the followUpActions(String reqId) method.
2. Define plugin.xml in following standard format, as specified by the plug-in framework:

```
<oimplugins xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <plugins pluginpoint="oracle.iam.request.plugins.StatusChangeEvent">
        <plugin pluginclass="com.mycompany.RequestFailedChangeEvent"
```

```
version="1.0" name="RequestFailedChangeEvent">
            <metadata name="status">
                <value>Request Failed</value>
            </metadata>
         </plugin>
</oimplugins>
```

In this XML definition, the metadata part specifies at which stage the plug-in must be run. This is done by specifying the metadata value as `Request Failed`, which means that the com.mycompany.RequestFailedChangeEvent plug-in will run when a request moves to the Request Failed status.

3. Register the plug-in with Oracle Identity Manager. See "Registering Plug-ins" on page 17-7 for information about registering plug-ins in Oracle Identity Manager.

13.7.2 Validating Request Data

You can use the RequestDataValidator plug-in to add custom validation of request data after submission. The plug-in point for this is the **oracle.iam.request.plugins.RequestDataValidator** interface with public void validate(RequestData requesterData) method.

You can define the dataset validators and prepopulation adatpers associated with the given plug-in. The request datasets associated with the plug-ins can be defined at the time of plug-in registration. The plugin.xml file is used to define the association between plug-ins and dataset validator or prepopulation adapters. The <metadata> node attached with the <plugins> element is used to define the association between data validators and prepopulation adapters.

Note: DataSetValidator plug-in specified for a dataset cannot be overridden by the plug-in enhancement of specifying the validators metadata in the plugin.xml itself. For instance, the predefined dataset 'ModifyUserDataset' shipped with default validator does not get overridden by the custom implementation class. Therefore, the validator in dataset will be given precedence, currently there is no option to override it.

Example 13–1 shows how the plug-ins can be associated with data validators and prepopulation adapters.

Example 13–1 Associating plug-ins With Data Validators and Prepopulate Adapters

```
<?xml version="1.0" encoding="UTF-8"?>
<oimplugins xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <plugins pluginpoint="oracle.iam.request.plugins.RequestDataValidator">
     <plugin pluginclass=
"oracle.iam.plugin.appinst.ApplicationInstanceDataValidator" version="1.0"
name="AppInstDataValidator">
     <metadata name="DataValidator">
      <value>AppInstanceDataSet|ADAppDataSet|EBSDataSet</value>
   </metadata>
      </plugin>
  </plugins>
</oimplugins>
```

In this example, the following line of code indicates defining metadata xml element to indicate that the plug-in is associated with request data validator datasets:

```
<metadata name="DataValidator">
```

Note that `Attribute name="DataValidator"` in the metadata element indicates plug-in associated with request data validators.

Defining the names of the datasets to be associated with the current plug-in is indicated by the following line:

```
<value>AppInstanceDataSet|ADAppDataSet|EBSDataSet</value>
```

Note: Request dataset names must be delimited by the single pipe character (|).

Consider the following scenarios:

- Scenario I: Provisioning Users to a Target System
- Scenario II: Provisioning or Modifying Entitlement Request

13.7.2.1 Scenario I: Provisioning Users to a Target System

Suppose Oracle Identity Manager is configured for provisioning users to the AD User APAC target. A RequestDataValidator specifies ADUserDataValidator is configured for the corresponding request dataset, as shown:

```
<plugin pluginclass= "oracle.iam.plugin.appinst.ADUserDataValidator" version="1.0"
name="ADUserDataValidator">
<metadata name="DataValidator">
<value>ADUserAPACDataSet</value>
</metadata>
</plugin>
```

Later, if the System Configurator wants to configure Oracle Identity Manager for provisioning users to the AD User EMEA target, then the System Configurator would create a new application instance, and associate a UI form with it. Request dataset would be auto-generated in the process. If the data-validator is to be re-used for this request dataset, then perform the following:

1. Edit plugin.xml of the ADUserDataValidator.
2. In the <metadata> <value> subtag, add the name of the new request dataset separated by a delimiter. For example:

   ```
   <value>ADUserAPACDataSet|ADUserEMEADataSet</value>
   ```

3. Re-register the data-validator plug-in.

13.7.2.2 Scenario II: Provisioning or Modifying Entitlement Request

Entitlement data provided as part of Provision Entitlement and Modify Entitlement request can be validated by creating a dataset validator plug-in and specifying the following in the plug-in metadata (plugin.xml):

```
<metadata name="DataValidator">
      <value>EBSForm.UD_EBS_RESP</value>
</metadata>
```

Here, EBSForm is the name of the form associated with the application instance on which the account is provisioned, and UD_EBS_RESP is the name of the child form corresponding to the entitlement. UD_EBS_RESP identifies the entitlement type.

13.7.3 Prepopulation of an Attribute Value During Request Creation

Prepopulation plug-in is associated with an attribute reference or attribute in request dataset. This can be used to prepopulate an attribute value by running custom code during request creation. Requester can modify the value that is prepopulated if required.

The plug-in point for this is **oracle.iam.request.plugins.PrePopulationAdapter** with public Serializable prepopulate(RequestData requestData) method. Use this plug-in only for the following request types:

Provision Resource, Self-Request Resource, Create User, Self-Register User.

Defining metadata element to indicate that the plug-in is mapped to request data set attributes for filling up prepopulated data is indicated by the following line:

```
<metadata name="PrePopulationAdapater">
```

The association is defined by combining dataset name with attribute name in the following format:

```
<DATASET_NAME>::<ATTRIBUTE_NAME>
```

For example:

```
AppInstanceDataSet::First Name
```

Multiple attributes can be associated with the same prepopulation plug-in, where each association is separated by the single pipe character (|). For example:

```
<datasetname1>::<attribute1> |
<datasetname2>::<attributename2>|<datasetname3>::<attribute3>
```

The following is an example of prepopulation plug-in:

```
<plugins pluginpoint="oracle.iam.request.plugins.PrePopulationAdapter">
  <plugin pluginclass=
"oracle.iam.plugin.appinst.ApplicationInstancePrePopulateAdapter" version="1.0"
name="AppInstPrepopAdapter">
     <metadata name="PrePopulationAdapater">
   <value>
AppInstanceDataSet::First Name|ADAppDataSet::Last Name
  </value>
   </metadata>
   </plugin>
  </plugins>
```

Note: In addition to creating request datasets by using the catalog Form Designer, you can manually upload request datasets to MDS. You can also define DataSetValidator or PrepopulationAdapter elements within the request dataset. These dataset validators or prepoulation adapters configured in the dataset have the highest priority over other configuration.

For example, a plug-in EBSUserDataValidator is registered to associate it with a request dataset EBSUSerDataSet, but the dataset has not been created or uploaded. Another plug-in ADUserDataValidator is registered but not associated with any request dataset. When you later create the request dataset EBSUSerDataSet and use it for creating requests, the plug-in EBSUserDataValidator is called for validating the request data. Then, you add the DataSetValidator element to the request dataset EBSUSerDataSet that you manually created, and specify another plug-in ADUserDataValidator. When you use EBSUSerDataSet to create requests, the plug-in ADUserDataValidator is called. This is because ADUserDataValidator is configured as a part of the request dataset. If the DataSetValidator entry is removed from EBSUSerDataSet, then the plug-in EBSUserDataValidator is invoked to validate the request data.

13.8 Enabling Auto-Approval for Self Registration Requests

Rules in approval workflow policies can be configured that determine whether a request should be auto-approved or a SOA composite should be invoked. For information about configuring approval workflow rules, see "Managing Approval Workflows" in *Administering Oracle Identity Manager*.

After you configure rules in the approval workflow policies for auto-approval, perform the following steps to enable auto-approval for self registration requests:

1. To assign the organization automatically, configure a home organization policy. See "Managing Home Organization Policy" in Administering Oracle Identity Manager for information on how to configure home organization policies.
2. By default, the Role/User Type set for the self registration requests is Part-Time Employee. If you want to overwrite this value, then change the plug-in configured in the SelfCreateUserDataset.xml dataset.
3. You can create a new plug-in implementation for the value/logic required and change the plug-in configured in the dataset to bring the new one in affect. The new plug-in must implement oracle.iam.request.plugins.PrePopulationAdapter.
4. Register the plug-in that you created by using the Plugin Registration Utility. For details, see "Registering and Unregistering Plug-ins By Using the Plugin Registration Utility" on page 17-8.
5. To update the request dataset:
   a. Export the /metadata/iam-features-requestactions/model-data/SelfCreateUserDataset.xml request dataset from the MDS, as described in "Exporting Metadata Files to MDS" on page 24-1.
   b. Update the name of the plug-in configured for Role attribute, as shown:

```
<AttributeReference name="Role" attr-ref="Role" available-in-bulk="false"
type="String" length="255" widget="dropdown"
```

```
lookup-code="Lookup.Users.Role"
required="true">
<PrePopulationAdapter
classname="oracle.iam.selfservice.uself.uselfmgmt.plugins.RolePrepopulateAd
apter"
name="RolePrepopulateAdapter"/>
</AttributeReference>
```

c. Import the updated request dataset to MDS, as described in "Importing Metadata Files from MDS" on page 24-2.

Note: Dynamic Monitoring Service (DMS) can be used to view performance metrics. The following DMS metrics are present for monitoring the performance of Self Registration flow:

- `Self_Registration`: This provides the number of completed and failed self registration requests.
- `oracle.iam.selfservice.uself.uselfmgmt.api.UnauthenticatedSelfService`: This provides details, such as the number of self registration requests and time taken to submit a self registration request.

13.9 Hiding the Skip Current Assignment Option

Skipping the current assignment is not a valid action for an approver. If an approver chooses this action, then the corresponding request fails. Therefore, you can hide this option by performing any one of the following ways:

- Change the task actions from the SOA composer. For the Skip Current Assignment action, deselect all the checkboxes and save.
- Change the task action by using JDeveloper. For the Skip Current Assignment action, deselect all the checkboxes. Then save and redeploy the composite. See "Specifying Actions for Acting Upon Tasks" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite* for more information.

13.10 Customizing Certification Oversight

Certification oversight can be customized to extend the levels of oversight or stop the oversight process when a certain title is reached. The certification composite contains customizable oversight logic that supports queries to Oracle Identity Manager to select a sequence of overseers based on any one or all of the following:

- The primary reviewer
- The current phase of certification
- The management-hierarchy defined in Oracle Identity Manager

By default, only a single level of oversight is supported such that a certification task is assigned to one reviewer.

As predefined in the composite for certification oversight, whenever the primary reviewer or an overseer signs off, the primary-review task is automatically routed to the next overseer in the sequence. After a primary reviewer or an overseer, except the final overseer, has signed off on the primary review task, that user will no longer be able to view the task in the inbox by querying for completed tasks.

Customizing the certification oversight involves the following steps:

1. Create the composite. To do so:

   a. Go to the *OIM_HOME*/server/workflows/new-workflow/ directory. The process-template subdirectory contains the ZIP file archives with composite files that are used as the base files to create the new composite.

   b. Run the following command:

   ```
   ant -f new_project.xml compliance
   ```

   You are prompted to make the following selection:

   1 - Identity Audit Composite

   2 - Certification Composite [Default]

   c. Choose option 2 for certification composite.

   d. When prompted, enter a name for the new composite, and press `Enter`. The composite is created, and a package directory with the composite name that you specified is created in the process-template subdirectory.

2. Open the composite in JDeveloper. To do so:

   a. Go to the process-template directory.

   b. Go to the directory with the composite name provided in step 1c.

   c. Open *COMPOSITE_NAME*.jpr using JDeveloper.

3. In the Projects pane of the Application Navigator view, expand the project and edit the **CertificationTask.task** by double-clicking. Click the **Assignment** tab. Click **Stage1.Participant** object, and select **Edit**. The Edit Participant Type dialog box is displayed. By default, the composite defines a single level of certification, and the certifications will be assigned to a single reviewer. For example, to change the level of certification to go to the manager of the current task assignee, set the following values:

   Note: The customization described in this procedure is a sample. For further customizations, see the CertificationOverseerProcess default template, and/or refer to SOA documentation.

   - **Type:** Serial. Only serial certification logic is supported.
   - **Build a list of participants using:** Management Chain
   - **Specify attributes using:** Value-based
   - **Starting Participant:** Select a user from which the certification starts.
   - **Top Participant:** By Title. Specify a title for the reviewer who is the top participant of the certification review. If you specify VP as the top participant and 5 as the number of levels, then the certification will go up to VP level even though it is level 3. The other levels will be skipped.
   - **Number of Levels:** By Number. If you specify 1, then it means that certification will up to the manager's level. A value of 2 means that the certification will up to the manager's manager.
   - **Auto assign task to a single:** User
   - **Assignment Pattern:** Least Busy

4. Click **OK**.
5. Compile the composite, and deploy the composite JAR file to SOA by referring to SOA documentation.
6. Login to Oracle Identity System Administration, and create a certification definition by selecting the newly deployed composite in the Configuration page.

13.11 Customizing the Identity Audit Composite

To customize the identity audit composite:

1. Create the composite. To do so:
   a. Go to the *OIM_HOME*/server/workflows/new-workflow/ directory. The process-template subdirectory contains the ZIP file archives with composite files that are used as the base files to create the new composite.
   b. Run the following command:

      ```
      ant -f new_project.xml compliance
      ```

      You are prompted to make the following selection:

      1 - Identity Audit Composite

      2 - Certification Composite [Default]
   c. Choose option 1 for identity audit composite.
   d. When prompted, enter a name for the new composite, and press `Enter`. The composite is created, and a package directory with the composite name that you specified is created in the process-template subdirectory.
2. Open the composite in JDeveloper. To do so:
   a. Go to the process-template directory.
   b. Go to the directory with the composite name provided in step 1d.
   c. Open *COMPOSITE_NAME*.jpr using JDeveloper.
3. Make all the customizations for the IdentityAuditTask assignment by editing the `IdentityAuditRemediationTask.task`.
4. Ensure that the required callbacks are configured (Task Completion, Escalation, Expiry, Routing, ReAssignment, and Proxy). Look into Task Assignment callbacks from the newly created composite or the callbacks configured in default Identity Audit Remediation composite for reference.
5. Save the changes.
6. Compile the composite, and deploy the composite JAR file to SOA by referring to SOA documentation.
7. Login to Oracle Identity Self Service, and create a new identity audit scan definition by using the newly created composite from the identity audit configuration composite lookup.

Part IV

Data Synchronization

This part contains chapters that describe customizing reconciliation, using the Bulk Load Utility, and developing scheduled tasks.

It contains the following chapters:

14 Customizing Reconciliation

This chapter describes reconciliation features and architecture and the various aspects of customizing reconciliation operations in the following sections:

- Reconciliation Features
- Reconciliation Architecture
- Defining Reconciliation Rules
- Developing Reconciliation Scheduled Tasks
- Updating Reconciliation Profiles Manually
- Understanding Reconciliation APIs
- Postprocessing for Trusted Reconciliation
- Reconciliation FAQs
- Troubleshooting Reconciliation
- Populating Data in the RECON_EXCEPTIONS Table
- Reconciliation Best Practices
- Monitoring Reconciliation Performance Using DMS

14.1 Reconciliation Features

Reconciliation features can be divided into the following categories:

- Performance Enhancement Features
- Web-Based Event Management Interface
- Other Features

14.1.1 Performance Enhancement Features

The following features help increase performance during reconciliation:

- New Metadata Model - Profiles
- Parameters to Control Flow and Processing of Events
- Grouping of Events by Reconciliation Runs
- Grouping of Events by Batches
- Implementing Reconciliation Engine Logic in the Database

- Improved Java Engine
- Improved Database Schema

14.1.1.1 New Metadata Model - Profiles

If metadata is associated with a reconciliation target, then it limits the ability to run multiple jobs performing different types of reconciliation against the same target. Therefore, all configurations in various components of Oracle Identity Manager are stored centrally in an XML store called MDS.

For backward compatibility, current deployments continue managing their configurations through Oracle Identity Manager Design Console and the configuration continues to be stored in the Oracle Identity Manager database. The configuration APIs automatically read the configurations from the tables in Oracle Identity Manager and convert them into XML profiles, called default profiles, and associate those profiles with the existing reconciliation runs.

You manage all the metadata by using Oracle Identity Manager Design Console. Using Oracle Identity Manager Design Console, you can generate the default reconciliation profile. This can be used to regenerate the profile when reconciliation configurations are changed from Oracle Identity Manager Design Console. When configurations are imported from the Deployment Manager, the profile is generated by default.

All nondefault profiles can be completely managed by using any XML editor.

> **See Also:** "Reconciliation Profile" on page 14-8 for information about reconciliation profiles

14.1.1.2 Parameters to Control Flow and Processing of Events

This section consists of the following topics:

- Parameters to Control Event Processing
- System Property to Control AutoRetry

Parameters to Control Event Processing

BatchSize is the parameter to control event processing. This dictates the size of the batch. A batch size of 1 is equivalent to processing of events one at a time. Batch size is available as a system property and can be managed from Oracle Identity Manager Design Console. The property name is OIM.ReconBatchSize. The default value of the system BatchSize parameter is 500. For information about system properties, see "Managing System Properties" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager*.

System Property to Control AutoRetry

The Retry Count for recon event system property controls auto retry by indicating how many times an item needs to be retried before the reconciliation engine marks it as an error or sends it to manual queue. The value 0 for this property means that the auto retry option is not configured.

> **See Also:** "Handling of Race Conditions" on page 14-5 for more information about auto retry

14.1.1.3 Grouping of Events by Reconciliation Runs

All the events created in the reconciliation database are grouped by reconciliation runs. All events in a reconciliation run are grouped with a common reconciliation run

ID. Because each reconciliation run is associated with a profile, all events in a reconciliation run are processed by using the same profile. This helps in optimizing the performance because the configurations have to be retrieved only once per reconciliation run.

Each profile can use a different batch size. This enhances system performance for each target reconciliation by tuning the appropriate batch for it.

14.1.1.4 Grouping of Events by Batches

Batches are introduced to increase system performance during reconciliation. A batch consists of a number of events. It is a unit of processing in the reconciliation engine. The size of the batch is configurable. Reconciliation runs are broken into fixed size batches. For example, if a reconciliation run consists of 9900 events and batch size is 1000, then that reconciliation run is divided into 10 batches each with size 1000, and last batch with size 900.

Processing a batch as a unit optimizes system performance by eliminating the overhead of processing one event at a time. This also allows performing bulk operations wherever possible. Batches can also run in parallel to balance the use of hardware resources.

14.1.1.5 Implementing Reconciliation Engine Logic in the Database

In earlier releases, all engine logic was implemented in Java and the processing happened one event at a time. In 11*g* Release 2 (11.1.2.3.0), most of the logic to process the events is implemented as stored procedures. A combination for processing at batch level and the logic being implemented in PLSQL makes it possible to perform bulk operations at the SQL layer. The following steps are performed in bulk (one batch at a time):

- Required data check
- Applying matching rules
- Applying action rules

14.1.1.6 Improved Java Engine

Processing that cannot be performed in stored procedures and must be performed in Java layer also provides better performance than earlier releases of the engine for the following reasons:

- Java engine performs bulk operations by default:
  - Submits events in batches to the database
  - Submits bulk postprocess orchestration depending on the action
- Performs bulk operations wherever possible.

14.1.1.7 Improved Database Schema

A notable performance enhancement from the new database schema in 11*g* Release 2 (11.1.2.3.0) is by using horizontal tables for storing event details for various targets instead of using a single vertical table for storing the event details from various targets. A horizontal table is used for each profile.

> **See Also:** "Staging Tables" on page 14-4 for more information about horizontal tables

14.1.2 Web-Based Event Management Interface

Oracle Identity Manager provides a Web-based event management interface that allows you to manage the events from the Web. Authorized users are able to search for events, users, and handle exceptions by linking events with users and accounts. You can also close events, force failed events to be re-evaluated, and perform ad-hoc linking.

Ad-hoc linking refers to the ability provided to authorized users of the Event Management section to link an event to any user in Oracle Identity Manager. Although the reconciliation engine finds user matches for events, the user through this ad-hoc link feature can ignore those matches and select a different user. This allows you to handle exceptions resulting from error matches.

See Also: "Managing Reconciliation" in *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager* for information about the tasks performed to manage reconciliation events

14.1.3 Other Features

Other reconciliation features are described in the following sections:

- Staging Tables
- Handling of Race Conditions
- Ad Hoc Linking

14.1.3.1 Staging Tables

In earlier releases of Oracle Identity Manager, the reconciliation schema has one table to store all the event details from various targets. The list of attributes and their names and types that the various reconciliation events contain can vary from target to target. This means that events from one target can contain a different set of data compared to events from another target. The only way to store data from such events in a single table is by storing one attribute per row. Therefore, in earlier releases, each row in the event detail table represents a single attribute of reconciliation event data. For each attribute, it stores the event to which it belongs, the attribute name, type, and value. This is also referred to as vertical table in this document. Although vertical tables are beneficial from the point of view of flexibility and extensibility, it is not an efficient way to store event records from the performance prospective.

Storage in vertical tables is replaced by separate tables for each target, called horizontal tables or staging tables. They are called horizontal tables because instead of storing attributes of an event vertically in the table as rows (as many rows as there are number of attributes), the attributes of an event are stored as columns. This means that there are as many columns as there are number of attributes for a target. Each event is stored as a row. Because different targets can have different sets of attributes, each target has a separate table in the reconciliation schema to store event details. There can be multiple tables per target because of requirements to handle multi-valued attributes that are stored as rows in child tables.

Each row of the event detail table for a specific profile stores the list of reconciliation fields for a single event. For example, for trusted user reconciliation in which firstname, lastname, email attributes are being reconciled, there is the RA_XELLERATE_USER staging table with the following columns:

RE_KEY, RECON_FIRSTNAME, RECON_LASTNAME, RECON_EMAI

Creating and Maintaining Staging Tables

Staging tables can be created only when a target is being deployed against Oracle Identity Manager. This is because, at the time of target deployment, the reconciliation system knows the list of attributes and their types for the target, which needs to be reconciled.

Staging tables are updated when configurations are imported from the Deployment Manager or changes are made by using Oracle Identity Manager Design Console. To generate a staging table from Oracle Identity Manager Design Console, in the Object Reconciliation form, click **Generate Reconciliation Profile**.

14.1.3.2 Handling of Race Conditions

In earlier releases of Oracle Identity Manager, when an event is being reconciled, the reconciliation engine may not be able to process it successfully because before this event can be reconciled, another event needs to be reconciled. For example, before the reconciliation engine can reconcile an event that is supposed to create an account, the engine needs to reconcile an event that is supposed to create a user. This is called a race condition.

In Oracle Identity Manager 11*g* Release 2 (11.1.2.3.0), the race conditions are handled by setting the value of the 'Retry Count for recon event' system property. To configure auto retry, specify a value greater than 0 for this property. If you do not want to configure auto retry, then specify 0 as the value of the Retry Count for recon event system property.

When auto retry is configured, the reconciliation engine checks for the race conditions. If a race condition is found, then the reconciliation engine puts the reconciliation event in a re-evaluate queue until the retry count is exhausted.

A Reconciliation Retry Scheduled Task periodically checks if there is any event waiting for retry and is ready to be re-evaluated and if yes, it queues them up for reconciliation engine processing. This scheduled task is configured by default.

Note: If the auto retry count is exhausted, the reconciliation engine does not further process the event and sets the status per the matching rules. However, you can manually retry by requesting for re-evaluate from Event Management. For information about re-evaluating events, see "Re-evaluating Events" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager*.

Auto retry can handle the following race conditions:

- An account event for creating an account in Oracle Identity Manager is processed before the user is created for this event because the event for creating user is not processed yet.
- A user event for creating a Xellerate user in Oracle Identity Manager is processed before the organization is created to which this user belongs.

All auto retry parameters are stored as part of the reconciliation profiles. This means that while the events belonging to one reconciliation run may have auto retry configured, the events belonging to another reconciliation run may not have auto retry configured.

In Oracle Identity Manager, there is no UI to manage these parameters within a profile and you must use an XML editor to manage them by directly editing the XML profile. For information about editing an XML profile, see "Creating and Updating

Reconciliation Profiles" on page 14-24.

14.1.3.3 Ad Hoc Linking

If the reconciliation engine is not able to determine the owner based on the matching rules, then you can manually link an account to a user by using Oracle Identity Manager Advanced Administration. Subsequent modifications to the account is automatically linked to that account.

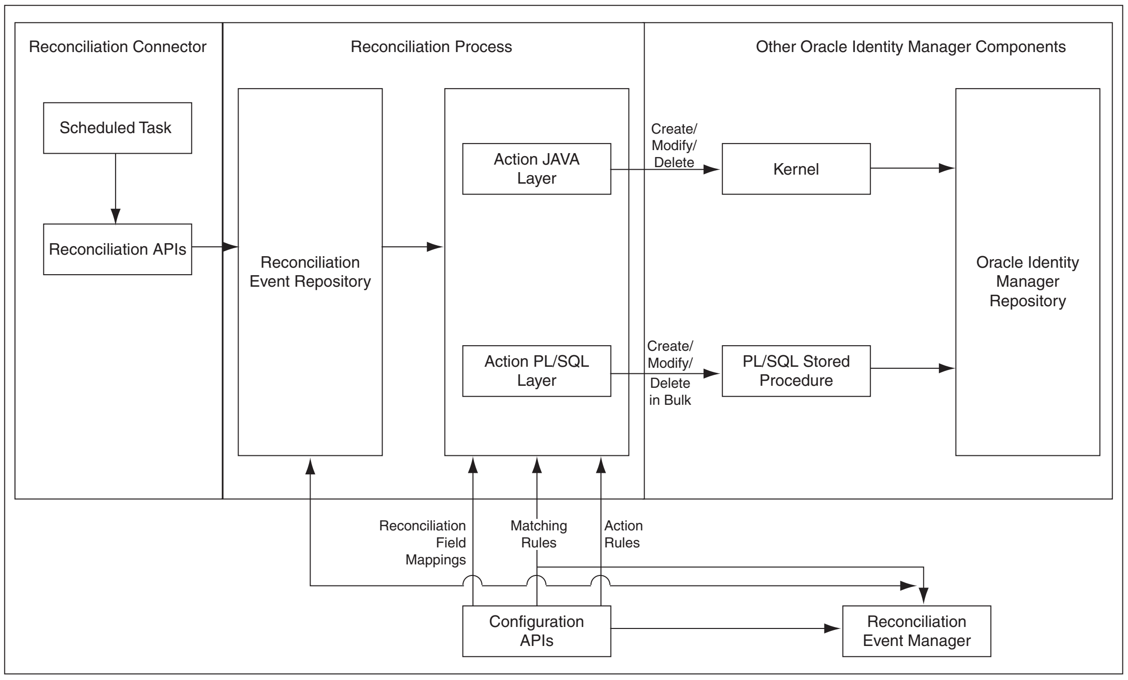
Ad hoc linking is supported for user and account events. If the reconciliation engine is not able to determine the owner based on the matching rules, then you can manually link a user or account event to a user.

> **See Also:** "Ad Hoc Linking" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager* for information about how to perform ad hoc linking

14.2 Reconciliation Architecture

Reconciliation is the process of pulling entity data from the target system into Oracle Identity Manager to keep the entity data in a consistent state between the two systems. The various components of Oracle Identity Manager involved in reconciliation and the interaction between these components are shown in the Figure 14–1:

Figure 14–1 Reconciliation Architecture



The reconciliation architecture is described in the following steps:

1. Each connector has scheduled tasks associated with it. The scheduler triggers the connector scheduled task, which invokes reconciliation APIs to generate events. The event can be of type Regular, Changelog, or Delete.

   For more information about the scheduler, see "Managing the Scheduler" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager*. For more information about scheduled tasks, see "Connector for Reconciliation" on page 14-16.

2. The reconciliation events are stored in the reconciliation event repository, which is Oracle Identity Manager database.
3. When batch size is met, an asynchronous message is submitted which processes the batch of events in bulk. At the end of the schedule task another asynchronous message is submitted for processing the events of the last batch.

Note:

- In Figure 14–1, the reconciliation engine encapsulates the Action JAVA Layer as well as parts of the Reconciliation Event Repository, and orchestrates all the arrows in that diagram.
- In this release, trusted source reconciliation is supported for users only. It is not supported for roles, role membership, and role hierarchy reconciliation.
- In this release, Oracle Identity Manager supports trusted source reconciliation and account reconciliation for organizations.

---

4. The processing involves data validation, matching of the entities and action (create, update, delete and so on). This is followed by post processing via kernel orchestrations. For information about the action module, see "Action Module" on page 14-15. For information about the reconciliation profile, see "New Metadata Model - Profiles" on page 14-2.
5. By default the reconciliation event processing happens in bulk, and therefore all the steps till post processing are performed by PL/SQL stored procedures. Event can be processed one at a time in the following scenarios (in this case all the steps till matching are done in PL/SQL and the action is performed in java layer):
   - When events are processed from the Event Management UI
   - When failed events are retried by the retry scheduled task that runs periodically

   For reconciliation single event processing, actions and post processing take place through the kernel.
6. Reconciliation events are made available to the Event Management UI by another API call in the reconciliation management service.

The functionality of various components of the reconciliation service are explained in the following sections:

- Reconciliation Profile
- Reconciliation Metadata
- Reconciliation Target
- Reconciliation Run
- Reconciliation APIs
- Reconciliation Schema
- Reconciliation Engine
- Connector for Reconciliation
- Archival

- Backward Compatibility
- Reconciliation Event Management

14.2.1 Reconciliation Profile

A reconciliation profile is the configuration defined to govern how reconciliation is run for a particular resource. A particular resource can have multiple reconciliation profiles, each of which defines matching rules, action rules, and field mappings, which can differ in each profile corresponding to the resource. For example, while one reconciliation run can perform reconciliation of new and modified accounts, another reconciliation run can reconcile deletion of accounts because you might want to run the deletions only once a day. In this example, you define two reconciliation runs and two profiles. Each profile is associated with respective reconciliation run and each profile having its own rules of reconciliation.

The profile is an XML-based configuration file stored in Oracle Identity Manager MetaData Store (MDS). Example 14–1 shows a sample reconciliation profile:

Example 14–1 Sample Reconciliation Profile

```
<?xml version='1.0' encoding='UTF-8'?>
<profile xmlns="http://www.oracle.com/oracle/iam/reconciliation/config" ownerType="User"
changeType="CHANGELOG" auditEnabled="true" batchSize="500" resourceType="Account" name="Modified AD
User" configure="true" active="true">

<matchingRule>((UPPER(USR.usr_udf_obguid)=UPPER(RA_ADUSERE469E5C8.RA_OBJECTGUID)))</matchingRule>
   <form oimTableName="UD_ADUSER" stagingTableName="RA_ADUSERE469E5C8" name="Modified AD User"
mlsOimTable="mlsOIMTableIfAny" mlsStagingTable="mlsStagingTableIfmlsOIMTable">
      <matchingRule>(UD_ADUSER.UD_ADUSER_OBJECTGUID=RA_ADUSERE469E5C8.RA_OBJECTGUID)</matchingRule>
      <targetAttributes>
         <targetAttribute type="String" name="Status">
            <stagingField type="String" length="256" name="RA_STATUS"/>
         </targetAttribute>
 <targetAttribute type="String" name="copyStatus" ref="Status" mls="true">
            <stagingField type="String" length="256" name="COPY_STATUS"/>
            <oimAttribute type="String" fieldName="OIM_OBJECT_STATUS" fieldType="String"
name="OIM_OBJECT_STATUS"/>
         </targetAttribute>
         <targetAttribute type="String" name="password" encrypted="true" keyField="false"
required="false">
            <stagingField type="String" length="256" name="PASSWORD"/>
            <oimAttribute type="String" fieldName="UD_ADUSER_PASSWORD" fieldType="String" name="AD
Password"/>
         </targetAttribute>
         <targetAttribute type="Date" name="accountExpires">
            <stagingField type="Date" name="RA_ACCOUNTEXPIRES"/>
            <oimAttribute type="Date" fieldName="UD_ADUSER_DATE" fieldType="Date" name="Account
Expiration Date"/>
         </targetAttribute>
         <targetAttribute type="ITResource" name="IT Resource" keyField="false">
            <stagingField type="ITResource" length="19" name="RA_ITRESOURCE15641F83"/>
            <oimAttribute type="Number" fieldName="UD_ADUSER_AD" fieldType="Number" name="AD
Server"/>
         </targetAttribute>
 <targetAttribute type="String" keyField="true" name="objectGUID">
            <stagingField type="String" length="32" name="RA_OBJECTGUID"/>
            <oimAttribute type="String" fieldName="UD_ADUSER_OBJECTGUID" fieldType="String"
name="Object GUID"/>
         </targetAttribute>
      </targetAttributes>
      <form oimTableName="UD_ADUSRC" stagingTableName="RA_ADUSERGROUPDETA902DB909" name="memberOf">

<matchingRule>(UD_ADUSRC.UD_ADUSRC_GROUPNAME=RA_ADUSERGROUPDETA902DB909.RA_MEMBEROF)</matchingRule>
         <targetAttributes>
            <targetAttribute type="String" keyField="true" name="memberOf">
               <stagingField type="String" length="256" name="RA_MEMBEROF"/>
               <oimAttribute type="String" fieldName="UD_ADUSRC_GROUPNAME" fieldType="String"
name="UD_ADUSRC_GROUPNAME"/>
            </targetAttribute>
         </targetAttributes>
      </form>
   </form>
   <actionRules>
      <actionRule condition="One Entity Match Found" action="Establish Link"/>
   </actionRules>
</profile>
```

Table 14–1 describes the elements and the structure of the reconciliation profile XML file.

Table 14–1 Elements in the Reconciliation Profile XML

Element Level 1	Sub-element Level 2	Sub-element Level 3	Sub-element Level 4	Sub-element Level 5	Description
<profile>					The root element or object of the reconciliation configuration profile.
	<ownerType>				Populated only for role hierarchy, role membership, and account with values Role, Role, and User respectively.
	<changeType>				By default, or if the element is not present, then the value is CHANGELOG. Otherwise, the value can be REGULAR, CHANGELOG, or DELETE.
	<auditEnabled>				Used with account type profile only. By default or if the element does not exist, then value is false, and audit for the resource object is stopped.
	<batchSize>				Changes the size or number of reconciliation events per batch. By default, or if the element is not present, then batch size is 500.
	<resourceType>				Value can be any one of Account, User, Role, RoleRole, RoleUser, and Organization.
	<name>				This is the resource object name.
	<configure>				By default or if the element is not present, then the value is false. If reconciliation configuration is to be created or updated on a system, then this must be marked as true. After all manual corrections of a profile, this attribute must be marked as true. For test to production, mark this element as true before importing into target system.
	<active>				By default or if the element is not present, then the value is true. Value is false for corrupt or invalid profiles and marks profile unusable. Such profiles are never loaded into the system. After all manual corrections of a profile, this attribute must be removed or marked as true.

Table 14–1 (Cont.) Elements in the Reconciliation Profile XML

Element Level 1	Sub-element Level 2	Sub-element Level 3	Sub-element Level 4	Sub-element Level 5	Description
	<matchingRule>				Populated only for role hierarchy, role membership, and account with owner matching rule. Otherwise, the element is not present.
	<form>				This specifies one parent form per profile.
		<oimTableName>			Oracle Identity Manager table into which data will be reconciled.
		<stagingTableName>			Staging table into which data from the target system is stored before processing.
		<name>			Same as profile name for the parent form and same as multivalued attribute name for the child forms.
		<mlsOimTable>			Multilanguage supported (MLS) Oracle Identity Manager table into which data will be reconciled if resource object is MLA-enabled.
		<mlsStagingTable>			MLS staging table into which data from target system is stored before processing if resource object is MLS-enabled.
		<matchingRule>			Matching rule for the form (resource object associated with the profile), and is always required.
		<targetAttributes>			Groups all target attributes.
			<targetAttribute>		One for each attribute from the target system.
				<type>	Data type of the target attribute.
				<keyfield>	By default, the value is false. Used in matching rule for account resource type.
				<name>	Name of the attribute from the target system provided by the connector that starts reconciliation.
				<required>	If the attribute is required, then this element must be present.
				<encrypted>	If the value is true, then the attribute value will be encrypted and stored in staging and Oracle Identity Manager tables.
				<ref>	Name of the target attribute in the same form whose value will be copied and stored in this attribute.
				<stagingField>	Specifies the column of the staging table corresponding to the target attribute. This contains the following elements: **<type>:** data type of the staging table column. **<length>:** length/size of the staging table column/field. **<name>:** name of the staging table column.

Table 14–1 (Cont.) Elements in the Reconciliation Profile XML

Element Level 1	Sub-element Level 2	Sub-element Level 3	Sub-element Level 4	Sub-element Level 5	Description
				<oimAttribute>	Specifies the mapped Oracle Identity Manager domain attribute name. The element is present only if the target attribute is mapped. This contains the following elements: **<name>:** Oracle Identity Manager attribute name **<type>:** Oracle Identity Manager attribute type **<fieldName>:** Column name of the Oracle Identity Manager table corresponding to the Oracle Identity Manager mapped attribute **<fieldType>:** Column type of the Oracle Identity Manager table corresponding to the Oracle Identity Manager mapped attribute
		<form>			Specifies child form or forms for the parent or root form. It corresponds to a multivalued attribute.
			<matchingRule>		Matching rule for a child form.
			<targetAttributes>		This is the same element as the parent <targetAttributes> element. This element can be nested several times, for example, <form><targetAttributes><form><targetAttributes>.
	<actionRules>				Groups all action rules for the resource object.
		<actionRule>			An actionRule element for each action rule.
			<condition>		The value can be any one of No Matches Found, One Entity Match Found, Multiple Entity Matches Found, One Process Match Found, Multiple Process Matches Found.
			<action>		Can be anything based on the profile XSD.

There is always a default profile associated with reconciliation configurations for any resource object. The default profile can be explicitly generated from Oracle Identity Manager Design Console in the developer's environment or implicitly generated during import from the Deployment Manager. For details on how to create and update profiles, see "Updating Reconciliation Profiles Manually" on page 14-23.

14.2.2 Reconciliation Metadata

The reconciliation metadata consists of various configurations used in creating and processing the reconciliation events. The reconciliation metadata is stored in a logical container called a profile. For information about reconciliation profile, see "Reconciliation Profile" on page 14-8.

Examples of the reconciliation metadata are:

- **Mapping rules:** Used to map the data received from the target system to the data managed about that target system in Oracle Identity Manager.

- **Matching rules:** Used during the processing of each reconciliation event to correlate the event data to a particular account, user, or role in Oracle Identity Manager.
- **Action Rules:** Used to specify the actions taken by Oracle Identity Manager based on the result of the processing of a reconciliation event.
- **List of target attributes:** Used to define the data attributes received from the target system via reconciliation. It is used in the mapping rules, and is configured by using Oracle Identity Manager Design Console.

The various configurations used in creating and processing the reconciliation events are managed by using Oracle Identity Manager Design Console, and for backward compatibility, is stored in the same Oracle Identity Manager tables as in Oracle Identity Manager release 9.1.0. In addition, the configurations are also stored in the reconciliation profile.

Note: For reconciliation in Oracle Identity Manager, a metadata model is being used. See "Managing Reconciliation Events" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager*.

14.2.3 Reconciliation Target

Reconciliation target refers to an instance of an application that acts as a source of changes for Oracle Identity Manager. An example of reconciliation target is an HR system, which acts as a source of identities for Oracle Identity Manager. A reconciliation target can be a source of users or accounts.

14.2.4 Reconciliation Run

Reconciliation run refers to the combination of a reconciliation connector and associated configurations which when run by the scheduled task, performs the reconciliation based on the rules defined in the associated configurations. The scheduler runs reconciliation periodically at fixed intervals. Reconciliation runs are scheduled within Oracle Identity Manager scheduler to run at a specified frequency. All events created during a reconciliation run are grouped together by a unique reconciliation run ID.

14.2.5 Reconciliation APIs

These are a set of published APIs to provide reconciliation data to Oracle Identity Manager in the form of reconciliation events. Connectors can use the APIs to push data to the reconciliation event repository. Scheduled tasks can be setup to run the APIs when reconciliation is to be run on a scheduled basis. The existing connectors do not need to be changed because the existing APIs are supported.

14.2.6 Reconciliation Schema

The data that comes from the target system for reconciliation is stored in the reconciliation schema. The data contains the changes to be reconciled with Oracle Identity Manager.

Reconciliation schema refers to the set of schema tables to store the reconciliation data. The reconciliation schema is redesigned for performance reasons and future extensibility. See "Improved Database Schema" on page 14-3 for more information about the reconciliation schema.

14.2.7 Reconciliation Engine

The reconciliation engine uses all configurable components and includes the data processor and rule evaluator that use these components to convert input data into a list of action items. It also includes the components that determine whether or not the actions can be automated based on the rule context. When an action is performed, either automatically or manually, the engine performs the appropriate updates and provisioning actions.

The main task of the reconciliation engine is to perform the comparison, determine the action to be taken, and apply the action in Oracle Identity Manager. It contains two modules, which are described in the following sections:

- Matching Module
- Action Module

14.2.7.1 Matching Module

The matching rule specified in the profile is used to identify whether the record being searched, exists in Oracle Identity Manager or not. Matching rules are rules to identify whether the data is for an identity that Oracle Identity Manager already has a record of, or to identify the owner of the account in Oracle Identity Manager.

For account entities, when no record is found, an owner match is then performed to identify the owner of the account.

For role hierarchy events, matching is performed to identify the parent and child role.

Note: While performing role hierarchy and role membership reconciliation, the matching criteria must contain both Namespace and Role Name in the matching criteria. The following is an example of a matching rule:

((UGP.ugp_rolename=x) and (UGP.ugp_namespace=y))

Here, x is the name of the staging table name column that is mapped to Role Name, and y is the name of the staging column that is mapped to Namespace.

At the end of the evaluation, the match table contains all the possible matches found within Oracle Identity Manager that meet the criteria for the event, and the state of the event is updated to one of the statuses listed in Table 14–2:

Table 14–2 Reconciliation Status Events

Status Events	Description
Data Received	Event data has been created in the database and is ready for further processing.
Event Received	A reconciliation event has been created and is ready for further processing. The finishReconciliationEvent API has not yet been called.
Data Validation Failed	The reconciliation event record is invalid. For example, a role event with an invalid role category will fail to validate. This situation could indicate a race condition.
Data Validation Succeeded	The event data was successfully validated and the event can now safely be processed by the Engine.

Table 14–2 (Cont.) Reconciliation Status Events

Status Events	Description
Multiple Accounts Match Found	Given the current matching rules, multiple matching account records were found for the data.
No Account Match Found	Given the current matching rules, no matching account records were found for the data.
Single Account Match Found	Given the current matching rules, one matching account record was found for the data.
Multiple Org Matches Found	Given the current matching rules, multiple matching organization records were found for the data.
No Org Match Found	Given the current matching rules, no matching organization records were found for the data.
Single Org Match Found	Given the current matching rules, one matching organization record was found for the data.
Multiple Role Grants Match Found	Multiple matching records for user membership within a role were found.
No Role Grant Match Found	No matching records for user membership within a role were found.
Single Role Grant Match Found	One matching record for user membership within a role was found.
Multiple Roles Match Found	Given the current matching rules, multiple matching role records were found for the data.
No Role Match Found	Given the current matching rules, no matching role records were found for the data.
Single Role Match Found	Given the current matching rules, one matching role record was found for the data.
No Role Members Found	The Reconciliation Engine did not find role members matching the data, given the current matching rules.
No Role Parent Found	The Reconciliation Engine did not find a role matching the data, given the current matching rules.
Multiple Role Relationships Match Found	Given the current matching rules, reconciliation has found multiple role-to-role relationships that match data in the event.
No Role Relationship Match Found	Given the current matching rules, reconciliation did not find any role-to-role relationships that match data in the event.
Single Role Relationship Match Found	Given the current matching rules, reconciliation has found one role-to-role relationship that matches data in the event.
Multiple Users Match Found	Given the current matching rules, multiple matching user records were found for the data.
No User Match Found	Given the current matching rules, no matching user records were found for the data.
Single User Match Found	Given the current matching rules, one matching user record was found for the data.
Invalid Event Data Passed	The event contains invalid data.
Being Re-evaluated	The reconciliation event is being re-evaluated from the reconciliation event management UI.
Being Re-tried	The reconciliation event is being retried automatically. This status event has been deprecated.
Creation Failed	The user/account/role entity was not created successfully.

Table 14–2 (Cont.) Reconciliation Status Events

Status Events	Description
Creation Succeeded	The user/account/role entity was created successfully.
Delete Failed	The user/account/role entity was not successfully deleted.
Delete Succeeded	The user/account/role entity was deleted successfully.
Event Closed	The reconciliation event was closed from the reconciliation event management UI. The change is complete.
Update Failed	The user/account/role entity was not updated successfully.
Update Succeeded	The user/account/role entity was updated successfully.

14.2.7.2 Action Module

This module applies the action based on the event state, entity type, and the action rules, as listed in Table 14–3:

Table 14–3 Action Rules

Event State	Entity Type	Action	Description
No User Match Found	User	None	Does not perform any action
		Create User	Creates a user in Oracle Identity Manager
No Account Match Found	Account	None	Does not perform any action
User Matched	User or Account	None	Does not perform any action
	User	Establish Link	Modifies or deletes the matched user based on the change type
	Account	Establish Link	Owner identified - creates an account
Users Matched	User or Account	None	Does not perform any action
Account Matched	Account	None	Does not perform an action
		Establish Link	Modifies or revokes the account based on the change type
Accounts Matched		None	Does not perform any action
No Role Match Found	Role	None	Does not perform any action
Single Role Match Found	Role	None	Does not perform an action
		Establish Link	Modify or delete a role
	Role Membership	Create role membership	Grant a role member to Oracle Identity Manager
		Delete role membership	Delete a role member from Oracle Identity Manager
		None	Does not perform an action
	Role Hierarchy	Create role hierarchy	Creates a role hierarchy in Oracle Identity Manager
		Delete role hierarchy	Delete a role hierarchy in Oracle Identity Manager

Table 14–3 (Cont.) Action Rules

Event State	Entity Type	Action	Description
		None	Does not perform an action
Multiple Roles Matched	Role, Role membership and Role Hierarchy	None	Does not perform an action
No Role Grant Match Found	Role Membership	None	Does not perform an action
		Create Role Member	Creates a role member in Oracle Identity Manager
Single Role Grant Match Found	Role Membership	None	Does not perform an action
		Establish Link	Delete role member
Multiple Role Grant Match Found	Role Membership	None	Does not perform an action **Note:** This state does not occur because the role grant match is done by looking for the primary key, which is a combination of the usr key and the group key.
No Role Parent Match Found	Role Hierarchy	None	Does not perform an action
		Create role parent	Create a role parent in Oracle Identity Manager
Single Role Parent Match Found	Role Hierarchy	None	Does not perform an action
		Establish Link	Delete role parent
Multiple Role Parent Match Found	Role Hierarchy	None	Does not perform an action
Data Validation Failed	Role, Role Hierarchy, Role Member	Race condition	Does not perform an action. The event needs to be re-evaluated.
Parent role not found	Role Hierarchy	Race condition	Does not perform an action. The event needs to be re-evaluated.
Role member not found	Role membership	Race condition	Does not perform an action. The event needs to be re-evaluated.

14.2.8 Connector for Reconciliation

The connector refers to the software that extracts the changes from the target system and creates events in the reconciliation schema by calling the reconciliation APIs. If the connector that you want to use is shipped with a predefined reconciliation module, then a scheduled task definition is available. You use this component to control the frequency at which the target system is polled for changes to track data and other connector-specific parameters.

The connector for reconciliation is deployed by using the Deployment Manager. When the connector is deployed, the corresponding reconciliation profile for that connector is created in the metadata store (MDS), and horizontal tables that store the event data are also created.

Note: Do not manually update reconciliation profile or update any reconciliation configurations from the Deployment Manager or Oracle Identity Manager Design Console when a reconciliation run is still in progress. This is because, if a reconciliation field is deleted or updated when a reconciliation run is in progress, then the event data might not be valid any more.

For information about configuring connectors, see Oracle Identity Manager Connector documentation.

See Also:

- "Reconciliation Metadata" on page 14-11 for information about MDS
- "Staging Tables" on page 14-4 for information about the staging tables

14.2.9 Archival

The Reconciliation Archival utility allows you to move processed events from the active reconciliation tables to archive tables. The events to move can be selected based on a time range. Only linked and closed events, which means successfully processed or closed by an administrator, can be archived.

See Also: "Using the Reconciliation Archival Utility" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager* for information about how to use the Reconciliation Archival utility

14.2.10 Backward Compatibility

You do not need to change the existing reconciliation configurations or scheduled tasks to leverage the new reconciliation service.

The existing configurations for reconciliation setup in earlier Oracle Identity Manager releases continues to function after upgrading to 11*g* Release 2 (11.1.2.3.0). As part of the upgrade, corresponding reconciliation event tables are created for each of the existing object types being reconciled.

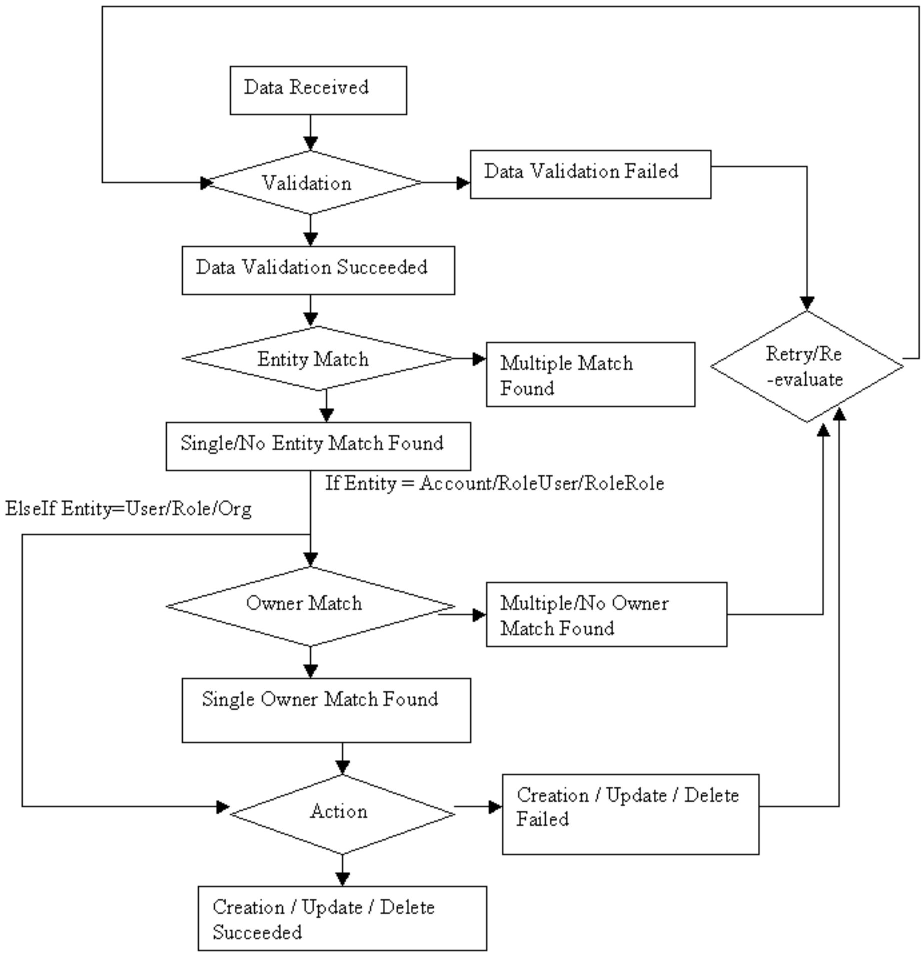
14.2.11 Reconciliation Event Management

The reconciliation events are managed by using the Event Management section of Oracle Identity System Administration. The Event Management section lets you view and manage reconciliation events generated by Oracle Identity Manager reconciliation engine. These events are generated through scheduled reconciliation runs. The Event Management section provides search capabilities on reconciliation runs as well as events. Users can use the Event Management section to perform reconciliation manually on generated events.

See Also: "Managing Reconciliation Events" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager* for more information about the managing reconciliation events in the Oracle Identity System Administration

Figure 14–2 shows the various stages in the lifecycle of a reconciliation event.

Figure 14–2 Reconciliation Event Lifecycle


Data Received
Validation
Data Validation Failed
Data Validation Succeeded
Entity Match
Multiple Match Found
Retry/Re -evaluate
Single/No Entity Match Found
If Entity = Account/RoleUser/RoleRole
ElseIf Entity=User/Role/Org
Owner Match
Multiple/No Owner Match Found
Single Owner Match Found
Action
Creation / Update / Delete Failed
Creation / Update / Delete Succeeded

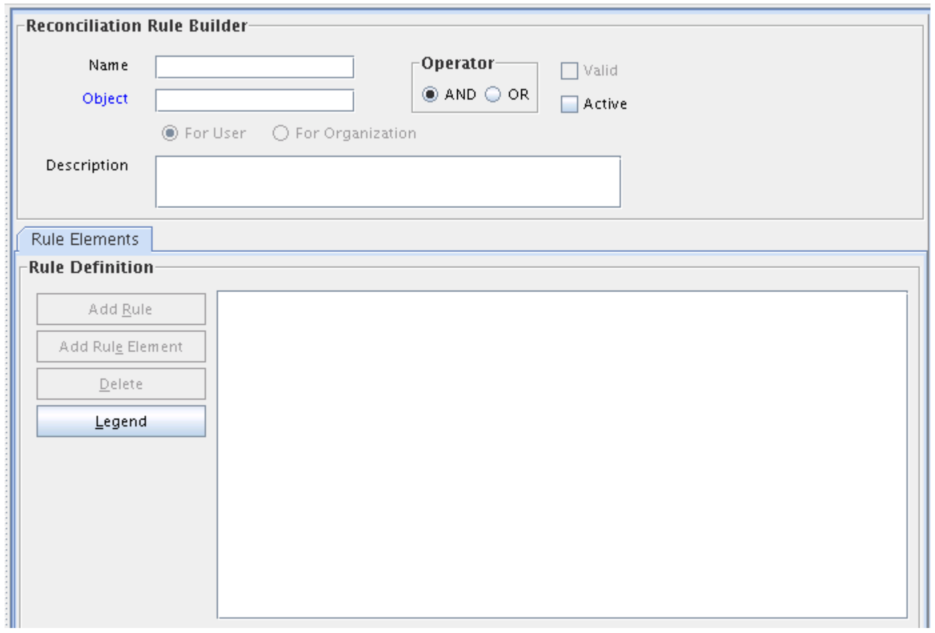
14.3 Defining Reconciliation Rules

You can define reconciliation rules that are invoked at the following instances:

- When Oracle Identity Manager tries to determine which user or organization record is associated with a change on a trusted source. These rules are evaluated as soon as all required fields in the reconciliation event are processed on the Reconciliation Data tab of the Reconciliation Manager form.
- When Oracle Identity Manager attempts to determine which user or organization record is the owner of an account discovered on a target resource, for example, as a result of a change detected on that system. These rules are evaluated only when all required fields in the reconciliation event are processed on the Reconciliation Data tab of the Reconciliation Manager form, and no processes were matched to the event on the Processes Matched Tree tab of the same form.

The Reconciliation Rules form in the Design Console is used to create and manage reconciliation rules in Oracle Identity Manager. This form is located in the Development Tools folder. Figure 14–3 shows the Reconciliation Rules form.

Figure 14–3 Reconciliation Rules Form



As mentioned, rules defined by using this form are used to match either users or organizations associated with a change on a trusted source or target resource. Rules of these types are referred to as user-matching or organization-matching rules, respectively. These rules are similar to the ones you can define by using the Rule Designer form except that the rules created by using the Reconciliation Rules form are specific to the resource object (because they relate to a single target resource) and only affect reconciliation-related functions.

Topics in working with reconciliation rules include:

- Defining a Reconciliation Rule
- Adding a Rule Element
- Nesting a Rule Within a Rule
- Deleting a Rule Element or Rule

14.3.1 Defining a Reconciliation Rule

The following procedure describes how to define a reconciliation rule.

Note: In the following procedure, you must ensure that the **Active** check box is selected. If this check box is not selected, the rule will not be evaluated by Oracle Identity Manager's reconciliation engine when processing reconciliation events related to the resource. However, you can only select this check box after Oracle Identity Manager has selected the **Valid** system check box. The **Valid** check box can only be selected after you have created at least one rule element, and Oracle Identity Manager has determined that the logic of this rule element is valid.

To define reconciliation rules for user or organization matching:

1. Go to the Reconciliation Rules form.

2. Enter a name for the rule in the **Name** field.
3. Select the target resource with which this rule is to be associated in the **Object** field
4. Enter a description for the rule in the **Description** field.

   Select the **And** or **Or** operator for the rule. If **And** is selected, all elements (and rules if they are nested) of the rule must be satisfied for the rule to be evaluated to true. If **Or** is selected, the rule will be evaluated to true if any element (or rule if one has been nested) of the rule is satisfied.
5. Click **Save**.

   The rule definition will be saved. Rule elements must now be created for the rule.

14.3.2 Adding a Rule Element

To define individual elements in a reconciliation rule:

1. Go to the Rule definition to which you want to add elements.
2. Click **Add Rule Element** on the **Rule Elements** tab.

   The Add Rule Element dialog box is displayed.
3. Click the **Rule Element** tab.
4. Select a user-related data item from the **User Data** menu.

   This will be the user data element that Oracle Identity Manager examines when evaluating the rule element. The menu will display all fields on the Oracle Users form (including any user-defined fields you have created).

   Note: If the rule being defined is for organization matching, both the data available and the name of the menus will be related to organizations, rather than users.

5. Select an operator from the **Operator** menu.

   This will be the criteria that Oracle Identity Manager applies to the attribute for data item you selected when evaluating the rule element. The following are valid operators:

   - **Equals**: If you select this option, the user or organization record's data element must exactly match the attribute you select.

     Note:

     - If you configure trusted source reconciliation of users, you must ensure that the `User ID` field of the Oracle Identity Manager User account is used in the reconciliation matching rule.
     - If you configure trusted source reconciliation of organizations, you must ensure that the `Organization Name` field of the Oracle Identity Manager User account is used in the reconciliation matching rule.

   - **Contains**: If you select this option, the user or organization record's data element must only contain (not be an exact match with) the attribute you select.

- **Start with**: If you select this option, the user or organization record's data element must begin with the attribute you select.
- **End with**: If you select this option, the user or organization record's data element must end with the attribute you select.

6. Select a value from the **Attribute** menu. The values in this menu are the fields that were defined on the Reconciliation Fields tab for the resource associated with the rule. If the reconciliation fields have not yet been designated for the resource, no values will be available.

   Note: When defining a rule element for a target resource (as opposed to a trusted source), only fields associated with parent tables of the resource's custom process form are available for selection in the **Attribute** field.

   ---

7. If you want Oracle Identity Manager to perform a particular transformation on the data in the **Attribute** field (before applying the operator), select the desired transformation from the **Transform** menu.

   Note: If you select a value other than None from this menu, after you click **Save**, you must also select the tab and set the appropriate properties so that Oracle Identity Manager is able to perform the transformation correctly.

   The possible transformations are described in Table 14–4.

Table 14–4 Transformation Properties

Transformation	Properties to Be Set on the Rule Element Properties tab
Substring	Start Point, End Point
Endstring	Start Point
Tokenize	Delimiters, Token Number, Space Delimiter

8. Select the **Case-Sensitive** check box.

   For the rule element to be met, if this check box is selected, the value selected in the **Attribute** field must match the capitalization of the value being evaluated in the reconciliation event record. If this check box is deselected, the value selected in the **Attribute** field is not required to match the capitalization used in the value being evaluated in the reconciliation event record.

9. Click **Save**.
10. If you select a value (other than None) in the **Transform** menu and have not yet set the properties for the transformation, the Properties Set check box will not be selected.

    You must select the **Rule Element Properties** tab, set the appropriate properties, and click **Save** again.

    The rule element will be added to the rule.

11. Repeat this entire procedure for each rule element you wish to add to the rule.

Note: Ensure that the **Active** check box is selected.

14.3.3 Nesting a Rule Within a Rule

You can nest an existing rule within a rule. Oracle Identity Manager evaluates the criteria of the nested rule in the same way as any other element of the rule.

Note: Only reconciliation-related rules that are associated with the same resource object are available for selection in the dialog box.

To nest a rule within a rule:

1. Go to the rule to which you want to add another rule.
2. Click **Add Rule** on the **Rule Elements** tab.
3. The Rule Choice lookup dialog box is displayed.

   Locate and select the desired rule.
4. Click **OK**.

   The selected reconciliation rule is added to rule.
5. Repeat steps 2 through 4 for each rule you want to nest in the rule.

14.3.4 Deleting a Rule Element or Rule

To delete a rule element or a rule:

1. Go to the rule from which you want to delete an element.
2. Select the rule element or rule to be deleted on the **Rule Elements** tab.
3. Click **Delete**.

14.4 Developing Reconciliation Scheduled Tasks

Oracle Identity Manager provides connectors for reconciliation of users/accounts from various target systems, such as Microsoft Active Directory, Sun Java System Directory, Oracle Internet Directory, and Oracle E-Business Suite. For information about these connectors, see Oracle Identity Manager Connectors Documentation in the Oracle Technology Network (OTN) Web site at the following URL:

`http://www.oracle.com/technetwork/indexes/documentation/index.html`

However, to create a custom connector, you must develop a new scheduled task that performs the following:

1. Retrieve user/account information from the target system.
2. Use reconciliation APIs to create reconciliation events to submit event data.
3. Create events for creating, modifying, or deleting an entity.

See Also: Chapter 16, "Developing Scheduled Tasks" for information about developing a scheduled task

To connect to a specific target system, you must:

- Create a new IT resource type

- Define a new IT resource
- Use the IT resource as an input parameter for the scheduled task

> **See Also:** *Oracle Fusion Middleware Java API Reference for Oracle Identity Manager* for information about the APIs to lookup IT resource definition

In Oracle Identity Manager, a provisioning process and a process instance is associated with activities related to users or accounts. This provides a hook or point to add customizations upon various actions.

Changes to the user state or the account state can occur via direct APIs or reconciliation. The changes can be of many types, such as:

> **See Also:** "Understanding Reconciliation APIs" on page 14-25 for information about the reconciliation APIs

- Data change in the user or account profile
- Status change, such as enable or disable
- Organization change
- Attribute propagation
- Password propagation

For each of these changes, the process definition provides a facility to add hooks to be run upon any of these changes. For reconciliation, the process definition provides the hooks in the form of the following conditional tasks:

- Reconciliation Insert Received: This conditional task is inserted when an account is created via reconciliation.
- Reconciliation Update Received: This conditional task is inserted when an existing account linked to a user is updated via reconciliation. Data in the process form or status of the account are updated.
- Reconciliation Delete Received: This conditional task is inserted when an existing account is revoked via reconciliation.

These tasks provide starting points for the workflows. You can create custom workflows in the provisioning process, and create a dependency between the reconciliation trigger tasks and the workflows. This causes the workflows to be run upon the respective triggers.

Every reconciliation event that is successfully linked to a user or an account inserts a single trigger from the conditional tasks. All the data in the user profile and the account profile is available as context-sensitive data for any adapter that is attached to one of these dependant tasks.

> **See Also:** Part II, "Connectors" and Part III, "Workflows" for details about creating conditional tasks, adapters, and dependencies

14.5 Updating Reconciliation Profiles Manually

This section describes updating reconciliation profiles manually in the following sections:

- Creating and Updating Reconciliation Profiles

- Changing the Profile Mode

14.5.1 Creating and Updating Reconciliation Profiles

For reconciliation based on resource objects, the profile name is the same as that of the resource object. For example, if resource object name is testresource, then the default profile name is also testresource. The corresponding reconciliation staging table name is available in the profile. If the resource has Multi-Language Support (MLS) data, then the MLS staging and Oracle Identity Manager table names are also available in the profile. See Table 14–1, " Elements in the Reconciliation Profile XML" for information about the structure and the elements in the reconciliation profile.

If the resource object has child forms, then for each child form, the Oracle Identity Manager table name and staging table name are available in the profile. Each staging table has a corresponding entity definition XML file, the name is same as staging table name with dot xml extension (.xml), which is stored in the MDS.

To change a anything in a reconciliation profile, for instance attribute batch size, either the profile can be updated manually or by using the Design Console. To update a reconciliation profile:

Note: If a reconciliation profile is changed by using the Design Console, the reconciliation profile must be regenerated by clicking the **Create Reconciliation Profile** button in the Object Reconciliation tab of the Design Console.

1. Export the /db/PROFILE_NAME profile document from MDS.
2. Make changes in the XML file, for example, change the batch size value.
3. Set the value of the configure attribute to true. For information about this attribute, Table 14–1, " Elements in the Reconciliation Profile XML".
4. Import the updated profile into MDS. See "Migrating User Modifiable Metadata Files" on page 24-1 for information about exporting and importing metadata to and from MDS.

   This automatically updates the staging tables and the corresponding staging table entity definitions.

14.5.2 Changing the Profile Mode

You can use one of the following methods to change the profile mode property from CHANGELOG to REGULAR:

See Also: "Mode of Reconciliation" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager* for information about changelog and regular reconciliation modes

- Change the value of the changeType attribute in the profile, for example:

```
<profile xmlns="http://www.oracle.com/oracle/iam/reconciliation/config"
changeType="REGULAR" batchSize="500" resourceType="Organization"
name="Xellerate Organization">
```

- Change the attribute during event creation:

The event creation API, introduced in Oracle Identity Manager 11*g* Release 2 (11.1.2.3.0), contains three parameters. The first two parameters are same as those used in previous create event APIs. The third parameter can have attributes such as dateFormat, changeType, eventFinished, and actionDate. The following is the constructor for the third parameter:

```
EventAttributes(boolean eventFinished, java.lang.String dateFormat, ChangeType
changeType, java.util.Date actionDate)
```

See *Oracle Fusion Middleware Java API Reference for Oracle Identity Manager* for more information about the third parameter.

You can use this API to set the changeType as follows:

```
public long createReconciliationEvent(String objName, Map<String, Object>
inputData, EventAttributes eventAttribs);
```

Note: Using the API to set the changeType attribute overrides the value of the changeType attribute set in the profile.

14.6 Understanding Reconciliation APIs

The reconciliation APIs are a set of published APIs that can be used to create reconciliation events with single-valued and multi-valued attribute data and other features.

Reconciliation connector developers must use these APIs to push data to the reconciliation event repository.

See Also: Chapter 20, "Using APIs" for more information about using APIs in Oracle Identity Manager

Most of these APIs existed in earlier versions of Oracle Identity Manager. However, in 11*g* Release 2 (11.1.2.3.0), the implementation has changed and is based on the new reconciliation architecture introduced in the release.

Existing standard connectors also use these APIs; since the earlier APIs continue to be supported, no changes are necessary to those connectors.

`callingEndOfJobAPI` is the only new reconciliation API in 11*g* Release 2 (11.1.2.3.0).

Each run of a connector is known as a job. In 11*g* Release 2 (11.1.2.3.0), reconciliation events are submitted to the reconciliation engine in batches. At the end of a job, the scheduler (which executes the connector scheduled task) executes a listener, which in turn invokes the `callingEndOfJobAPI`. This API submits any open batch for processing to the reconciliation engine.

The API calls are similar for Multilanguage Supported (MLS) and non-MLS data. The connector passes in data to be reconciled as a HashMap. The difference is that if an attribute is MLS-enabled, then the key is the attribute name, while the value is another HashMap of MLS data. The keys of this MLS-specific HashMap are language codes, and the values are the corresponding locale-specific data obtained from target system. If there is any MLS data that does not have a locale defined with it in the target system, that data is passed with key "base" in the MLS input data HashMap.

14.6.1 The ReconOperationsService API

The APIs in oracle.iam.reconciliation.api.ReconOperationsService are required for the following tasks:

- Ignore Event
- Create Event (single/bulk)
- Process Event
- Deletion Detection

> **See Also:** *Oracle Fusion Middleware Java API Reference for Oracle Identity Manager* for details about the APIs in oracle.iam.reconciliation.api.ReconOperationsService

The preferred API and the order of invocation of these APIs is as follows:

1. **Ignore Event:** This is a way to prevent event creation and processing of target system data that already exists in Oracle Identity Manager. The API invocation is as follows:

   ```
   boolean ignoreEvent(String resourceObjectName, Map inputData, String
   dateFormat) throws tcObjectNotFoundException, tcAPIException
   ```

   This API is used to validate whether or not the reconciliation create event needs to be raised for the specified object. If this API returns true, then you can skip the event creation, which saves extra event creation in the database.

   Similar to the ignoreEvent API, the ignoreEventAttributeData method can be used to validate whether or not the reconciliation create event flow needs to be raised for single and multivalued data coming from the target system. In this release, only the account entity type has such data. The API is as shown:

   ```
   boolean ignoreEventAttributeData(String resourceObjectName, Map inputData,
   String multiValueFieldName, Map[] childDataList, String dateFormat) throws
   tcAPIException, tcObjectNotFoundException
   ```

   > **Note:** Either ignoreEvent or ignoreEventAttributeData must be invoked; both the APIs are not required to be invoked.

2. **Create Event:** This can happen via single event creation or bulk event creation APIs. This flow simply stores target system data in staging tables. The processing of this data asynchronously takes place later on.

   Create Event (Single): This consists of the following APIs:

   - Use the createReconciliationEvent method to provide the data for creating reconciliation events. If there is child or multivalued data, then set the value of the eventAttribs.eventFinished flag to false. Otherwise, set this value to true. It returns the eventId of the created Event.

     ```
     long createReconciliationEvent(String resourceObjectName, Map<String,
     Object> inputData, EventAttributes eventAttribs)
     ```

   - The child data is provided using the addMultiAttributeData method. If there is no child data or the eventAttribs.eventFinished flag is set to true, then this API must not be invoked.

```
long addMultiAttributeData(long plReconciliationEventKey, java.lang.String
psFieldName, java.util.Map poData) throws tcAPIException,
tcEventNotFoundException,
tcEventDataReceivedException,tcAttributeNotFoundException
```

Note: For better performance with bulk multivalued attributes or the data for multiple child records instead of a single child record, use the following API:

```
void addDirectBulkMultiAttributeData(long reconciliationEventKey,
long reconciliationAttributeKey, String tableFieldName, List
dataList,String dateFormat) throws tcAPIException,
tcEventNotFoundException, tcAttributeNotFoundException,
tcEventDataReceivedException,tcInvalidAttributeException
```

- The providingAllMultiAttributeData method specifies whether the multivalued data being provided is the entire list of data, or only changeset that has been added/updated. By default, the value of the pbFlag is false. If there is no child data or the eventAttribs.eventFinished flag is set to true, then this API must not be invoked.

  ```
  public void providingAllMultiAttributeData(long plReconciliationEventKey,
  String psFieldName, boolean pbFlag) throws tcAPIException;
  ```

- The finishReconciliationEvent method is used to mark the end of event creation flow. Particular event status is updated to Data Received, which means that all the data for the particular event, including the child data if any, has been provided. If the eventAttribs.eventFinished flag is set to true, then this API must not be invoked.

  ```
  void finishReconciliationEvent(long eventId) throws tcAPIException,
  tcEventNotFoundException, tcEventDataReceivedException
  ```

- The callingEndOfJobAPI method processes all the reconciliation batches in the job. For a scheduled job, this API is automatically called when the job ends. This API must be explicitly called for a nonscheduled job API invocation.

  ```
  void callingEndOfJobAPI() throws tcAPIException
  ```

Create Event (Bulk): This consists of the following API:

```
ReconciliationResult createReconciliationEvents(BatchAttributes batchAttribs,
InputData... input)
```

This is the bulk create API. It creates bulk reconciliation events for the data passed in input data. It accepts all the data including multivalued attributes, and submits it for processing as one batch if the size of data is less then or equals to the batch size. Otherwise, it submits the data in multiple batches. There is no need to call any other API after this.

3. **Process Event:** This is a way to force the backend processing of an already created event. The processReconciliationEvent(eventId) API is invoked after create event flow has finished and an already created event needs to be processed as well. This API processes only a particular event, it does not update the batch or job status. If batch status needs to be updated as well, then invoke the callingEndOfJobAPI API after this. Using this API is not recommended because it is synchronous and processes data one at a time, rather than in batch.

Note: If the processReconciliationEvent API is used for processing, then set the reconciliation batch size (batchSize parameter) to 0 in the reconciliation profile of the resource object. See Table 14–5 for more information about this step.

---

4. **Deletion Detection:** This is a way to delete extra data in Oracle Identity Manager that does not exist in the target system. This consists of the following APIs:

   - The provideDeletionDetectionData method takes the list of all the existing target system data for a resource object as input, and then returns a list of matching data found in Oracle Identity Manager.

     ```
     Set provideDeletionDetectionData(String resourceObjectName, Map[]
     inputData) throws tcAPIException, tcIDNotFoundException,
     tcMultipleMatchesFoundException
     ```

   - The getMissingAccounts method takes the list keys of already found data in Oracle Identity Manager, and returns a list of extra data that is in Oracle Identity Manager but not in the target system. It retrieves all keys from Oracle Identity Manager and compares them with the keys present in the set returned by the provideDeletionDetectionData method.

     ```
     Thor.API.tcResultSet getMissingAccounts(String objectName, Set
     accountsFound) throws tcAPIException, tcIDNotFoundException,
     tcDataNotProvidedException
     ```

   - The deleteDetectedAccounts method takes a list of data found only in Oracle Identity Manager as input, and invokes a delete type create reconciliation event API call, one at a time. The tcResultSet returned by the getMissingAccounts method is passed as parameter to this API.

     ```
     long[] deleteDetectedAccounts(Thor.API.tcResultSet poDetectedAccounts)
     throws tcAPIException, tcAPIException
     ```

14.6.2 Invoking Non-scheduled Task-Based Reconciliation in a Multithreaded Environment

Example 14–2 shows the sample code to invoke non-scheduled task-based reconciliation in a multithreaded environment:

Example 14–2 Invoking Non-scheduled Task-based Reconciliation in a Multithreaded Environment

```
public class UserNonSTBasedRecon{

     private AtomicInteger threadCount =new AtomicInteger(0);

public Long getRandomLong(int maxValue) {
        Random random = new Random();
        long token = random.nextInt(maxValue);
        return token;
    }
     @Test
     public void testCreateUsersUsingNonScheduleTaskConnectorWithThreads() throws
Exception {

         Thread t = new CreateEvent();
```

```
         t.start();
         Thread t2 = new CreateEvent();
         t2.start();

     while (true) {
             Thread.currentThread().sleep(5000);
             if (threadCount.get() == 2){
             OIMClient oimClient = null;
             ReconOperationsService reconServ =
oimClient.getService(ReconOperationsService.class);
                 reconServ.callingEndOfJobAPI();
                 break;
             }
         }
     }

     public class CreateEvent extends Thread {

         @Override
         public void run() {

         String ctxFactory = "weblogic.jndi.WLInitialContextFactory";
             OIMProfileReader reader = new OIMProfileReader();
             String appServerType = reader.getString("appserver.type");
             String hostName = reader.getString("weblogic.host");
             String port = reader.getString("weblogic.port");
             String serverURL = "t3://" + hostName + ":" + port;
             System.out.println("Server URL is : " + serverURL);
             System.out.println("Context Factory is : " + ctxFactory);
             Hashtable<String, String> env = new Hashtable<String, String>();
             env.put(OIMClient.JAVA_NAMING_PROVIDER_URL, serverURL);
             env.put(OIMClient.JAVA_NAMING_FACTORY_INITIAL, ctxFactory);

             OIMClient client = new OIMClient(env);
             String username = "xelsysadm";
             String password = "Welcome1";
             try {
                 client.login(username , password.toCharArray());
             } catch (LoginException e1) {
                 throw new SuperRuntimeException(e1.getMessage(), e1);
             }

             String uniq2 = getRandomLong(10000).toString();
             long jobId = getRandomLong(10000);
             ContextManager.setValue(Constants.JOB_HISTORY_ID, new
ContextAwareNumber(jobId));
             ContextManager.setValue(Constants.JOB_NAME_CONTEXT, new
ContextAwareString(jobId +""));
             ReconOperationsService recon;
             try {
                 recon = client.getService(ReconOperationsService.class);
                 int count = 50;
                 HashMap<String, String> hm = new HashMap<String, String>();
                 ArrayList<Long> eventKeys = new ArrayList<Long>();
                 for (int i = 0; i < count; i++) {
                     hm.put("UserLogin", uniq2 + "ThreadTest" + i);
                     hm.put("FirstName", uniq2 + "Thread" + i);
                     hm.put("lastname", "Test");
                     hm.put("Type", "End-User");
                     hm.put("OrganizationName", "Xellerate Users");
```

```
                    hm.put("EmpType", "Full-Time");
                    hm.put("Middlename", "MID");
                    System.out.println("Creating Recon event i ="+ i);
                    long rceKey = recon.createReconciliationEvent("Xellerate
User", hm, true);
                    eventKeys.add(rceKey);
                }
                assertEquals(count, eventKeys.size());
            } catch (Exception e) {
                throw new SuperRuntimeException(e.getMessage(), e);
            } finally {
                threadCount.set(threadCount.get()+1);
                ContextManager.popContext();
            }
        }
    }
 }
```

14.7 Postprocessing for Trusted Reconciliation

If the user login is not passed for trusted reconciliation, then the login handler generates the user login. The password is generated in postprocessing event handler. You can configure Oracle Identity Manager to send notification for the same.

Notification is sent only when the value of the Recon.SEND_NOTIFICATION system property is set to true. See "System Properties in Oracle Identity Manager" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager* for information about the Recon.SEND_NOTIFICATION system property.

In SSO disabled environment, for user creation via reconciliation, both the user login and password are generated in postprocess handlers and a single notification is sent for both user login and password.

In SSO enabled environment, because the password is not to be generated, if login is generated in postprocess handler, then notification is sent only for the user login.

14.8 Reconciliation FAQs

This section provides the following FAQs about reconciliation:

What should be performed after reconciliation configuration and before reconciliation run?

After all the reconciliation configuration are done either by importing a connector or via the Design Console, before triggering the scheduled job to start the reconciliation run:

1. Make sure that IT resource details are correct and you are able to connect to the target system.
2. Validate if all the reconciliation configurations are correct. This can be done by validating the reconciliation profile using the ProfileValidator mbean available on the Enterprise Manager. For information about various profile configuration issues, see "Troubleshooting Reconciliation Profile Configuration Failures" on page 14-37.

How to handle data issues while event generation?

To handle data issues while event generation (when reconciliation events are not getting generated):

- Enable INFO level logs, as described in "Troubleshooting General Reconciliation Issues" on page 14-32.
- Check INFO levels logs to see if createReconciliationEvent() is getting invoked, which confirms the call to reconciliation engine. Look for log: `createEvent Input Data`. This also prints the data being passed from the connector to the reconciliation engine.
- Analyze the logs to check if the event is ignored if the same data already exists in Oracle Identity Manager, thereby not generating the event again, which is expected behavior.
- If no logs are displayed related to createEvent and ignoreEvent, then the issue is most likely in connector area, even before call is made to the reconciliation engine.

How to handle processing failures?

To handle processing failures:

- Know if the connector being used is a listener-based connector (push-based connector) such as PSFT or RACF, or is it a pull-based connector. Knowing this is important as push and pull-based connectors follow different processing paths.
- Check if processing is happening via single event flow, which is end-to-end orchestration, or is it bulk processing using SPs. Single event processing happens mostly with push-based connectors, which invokes the `processReconciliationEvent()` API or when a failed event is retried/re-evaluated.
- As mentioned in "Troubleshooting Reconciliation" on page 14-32, check the exception if any in the RE_NOTE/RB_NOTE. Complete exception details can be further checked in the Oracle Identity Manager server logs.
- See "Troubleshooting Reconciliation" on page 14-32 for other common failures and resolutions during processing.

How to handle post-processing failures?

To handle post-processing failures:

- As mentioned in "Troubleshooting Reconciliation" on page 14-32, get the orchestration IDs from RB_NOTE/RE_NOTE columns for bulk and single event processing respectively.
- You can also validate the event by using the EventDiagnostic Mbean, which provides all details related to event processing along with post processing details, such as what is the orchestration status, which all handlers got executed, and failure details if any.
- If there is some custom event handler not getting executed only in reconciliation flow, then check if the bulk implementation exists or not. The orchestration-related info is available only in the logs and cannot be diagnosed by using above steps.

How to do performance-related analysis?

For performance-related analysis for reconciliation, see "Monitoring Reconciliation Performance Using DMS" on page 14-42. In addition, refer to the technote "Oracle Identity Manager 11g Reconciliation Performance Tuning" with Doc ID 1484808.1 at My Oracle Support web site at:

https://support.oracle.com

14.9 Troubleshooting Reconciliation

Before troubleshooting issues related to reconciliation, change the reconciliation logging level to INFO. To do so, add the following logger by using Oracle Enterprise Manager:

- **Name:** oracle.iam.reconciliation
- **Oracle Diagnostic Logging Level (Java Level):** NOTIFICATION:1(INFO)

For detailed steps of adding a logger, see "Configuring Logging" in *Administering Oracle Identity Manager*.

Note: To change the logging level, you can also modify the /domains/*DOMAIN_NAME*/config/fmwconfig/servers/*OIM_SERVER*/logging.xml file. To do so:

1. In the logging.xml file, add a new logger, as shown:

   ```
   <LOGGER NAME="oracle.iam.reconciliation" LEVEL="INFO"/>
   ```

2. Change the logging level of the 'console-handler' log_handler to INFO.
3. Restart Oracle Identity Manager.

This section describes troubleshooting reconciliation issues in the following sections:

- Troubleshooting General Reconciliation Issues
- Troubleshooting Database-Related Reconciliation Issues
- Troubleshooting Reconciliation Profile Configuration Failures
- Troubleshooting LDAP Reconciliation Issues

14.9.1 Troubleshooting General Reconciliation Issues

Table 14–5 lists the troubleshooting steps that you can perform if you encounter reconciliation errors:

Table 14–5 Troubleshooting Reconciliation

Problem	Solution
Failure in processing events	The error details can be obtained from the reconciliation tables, such as: ■ For batch processing, the exception is stored in RECON_BATCHES.RB_NOTE column ■ For single event processing, the exception is stored in RECON_EVENTS.RE_NOTE column

Table 14–5 (Cont.) Troubleshooting Reconciliation

Problem	Solution
Various data corruption issues resulting due to duplicate processing (both single and bulk processing) in case of push-based connectors when they are processing reconciliation using the processReconciliationEvent API For example, duplicate account creation, status unexpectedly getting changed, and so on.	Set the reconciliation batch size (batchSize parameter) to 0 in the reconciliation profile of the affected resource object.
Failure occurring in kernel orchestration handler	The orchestration ID can be tracked from the reconciliation table, which can further be used to check the status of related handlers, such as: ■ For batch processing, the postprocess only orchestration ID can be read from the RECON_BATCHES.RB_NOTE column ■ For single event processing, end-to-end orchestration ID can be read from the RECON_EVENTS.RE_NOTE column There is no UI that displays LDAP synchronization during reconciliation. Therefore, you can only track LDAP success or failure by checking the status of LDAP sync event handlers in orchestration based on the ID available in RB_NOTE/RE_NOTE columns.

After a job run, a lot of events are in the Data Received status.	Check if related batches are in Ready For Processing status by using the following statement:
	`select rb_batch_status, rb_note from recon_batches`
	`where rb_batch_status = 'Ready For Processing' and`
	`rj_key =` *`JOB_ID_ON_UI`*
	
	In addition, in the RECON_BATCHES.RB_NOTE, there is some generic exception, such as Connection issue. Fix the issue, and then perform any one of the following:
	■ If there is a lot of data, then rerun the reconciliation job.
	■ There is a scheduled task provided for manual retry of such failed batches Retry Reconciliation Batch. This can be used for retrying specific batches only. Multiple comma-separated batches are supported.
	■ There is no predefined job associated with this schedule task. A job can be created as required.

Race Condition - Events are in failed status because some dependent attribute is still not reconciled, for example, user's manager/organization needs to be reconciled before user.	■ If the size of the data is small, then retry reconciliation automatically handles the race condition, but it is slow. ■ If the size of the data is large, then perform the reconciliation two times. Remove the mapping for the dependent field for the first full reconciliation, and then add the dependent field mapping and perform the full reconciliation second time.

Table 14–5 (Cont.) Troubleshooting Reconciliation

Problem	Solution
The following error is generated when performing user update for trusted source reconciliation: `ORA Error Code =>ORA-00001: unique constraint (.) violated`	For of trusted source reconciliation, if the matching rule is based on Usr_login, then the matching rule must not be case-sensitive. Otherwise, updating users work as creating users, and the error might be generated.
Recon events skipped/not created error is generated with the following SQLException in the logs: `<oracle.iam.reconciliation.dao.event> <BEA-000000> <Generic Information: {0} java.sql.SQLException: ORA-12899: value too large for column "DEV_OIM"."RA_LDAPUSER9504ECC4"."RA_BUSINESSPHONE" (actual: 26, maximum: 20) at oracle.jdbc.driver.T4CTTIoer.processError(T4CTTIoer.java:462) at oracle.jdbc.driver.T4CTTIoer.processError(T4CTTIoer.java:405)`	This issue is because the data passed for the fields from the target is of greater size than the column size in the database table. To resolve this you should either: ■ Modify the data in the target for these fields, as per the column size in the database table, or ■ Modify the field length for these columns from the console and then recreate the recon profile

---

Note: The oracle.iam:type=Reconciliation,name=EventDiagnostic MBean with method 'diagnose' can be used to diagnose end-to-end reconciliation event flow. This MBean can be accessed by using Oracle Enterprise Manager. The diagnose method takes reconciliation EventID as input, and shows the following information about the event:

- Event, batch, job, and history details. This includes the RE_NOTE and the RB_NOTE values, and therefore, indicates the reason for failure, if any, or else the associated orchestration IDs.
- Old state table data, which is relevant for audit.
- Staging table data, which is data coming to reconciliation from the target system.
- Orchestration details, which includes all the event handlers that executed along with their status and reason for failure, if any.

14.9.2 Troubleshooting Database-Related Reconciliation Issues

This section the describes the following database-related issues for reconciliation:

Missing Critical Oracle Database 11*g* Release 1 Interim Patches

When the RDBMS interim patch# 7614692 is missing, the following error is logged:

```
ORA-02291: INTEGRITY CONSTRAINT (FK_RECON_EVENTS_USR) VIOLATED - PARENT KEY NOT
FOUND
[EXEC] ORA-06512: AT "OIM_SP_RECONBLKUSERCRUD"
[EXEC] ORA-06512: AT "OIM_SP_RECONBLKUSRMLSWRAPPER"
[EXEC] ORA-06512:
```

To resolve this issue, the following patches must be installed on Oracle Database 11*g* Release 1 (11.1.0.7.0):

- p7614692_111070
- p7000281_111070
- p8327137_111070
- p8617824_111070

Note: You can download all interim patches from the following URL:

http://support.oracle.com

Missing Critical Oracle Database 11*g* Release 2 Interim Patches

Running some SQL scripts might generate incorrect or inconsistent results on Oracle Database 11*g* Release 2 (11.2.0.2.0), which do not cause problems in earlier release of Oracle Database.

To resolve this issue, patch# 10259620 for Oracle Database 11g Release 2 must be installed.

Slow Reconciliation and Similar Traces in Error Log

When the SQL scripts having matching rules involving large volume, the entity tables are slow probably because of FULL table scans or unoptimized SQL plans in the database.

Reconciliation can be slow when the matching rule columns are not properly indexed or schema statistics is outdated. The slowness results in error logs similar to the following:

```
oracle.iam.platform.utils.SuperRuntimeException: java.sql.SQLException:
ORA-01013: user requested cancel of current operation
ORA-06512: at "XL_SP_RECONBLKROLEMATCH"
ORA-06512: at "OIM_SP_RECONBLKROLEMLSWRAPPER"
ORA-06512:

at weblogic.jms.client.JMSSession$UseForRunnable.run(JMSSession.java)
at weblogic.work.SelfTuningWorkManagerImpl$WorkAdapterImpl.run(SelfTuningWorkMana
gerImpl.java)
at weblogic.work.ExecuteThread.execute(ExecuteThread.java)
at weblogic.work.ExecuteThread.run(ExecuteThread.java)
Caused by: java.sql.SQLException: ORA-01013: user requested cancel of current
operation
ORA-06512: at "XL_SP_RECONBLKROLEMATCH"
ORA-06512: at "OIM_SP_RECONBLKROLEMLSWRAPPER"
ORA-06512:
.
at oracle.jdbc.driver.SQLStateMapping.newSQLException(SQLStateMapping.java)
at oracle.jdbc.driver.DatabaseError.newSQLException(DatabaseError.java)
at oracle.jdbc.driver.DatabaseError.throwSqlException(DatabaseError.java)
at oracle.jdbc.driver.T4CTTIoer.processError(T4CTTIoer.java)
at oracle.jdbc.driver.T4CTTIoer.processError(T4CTTIoer.java)
```

To resolve this issue:

1. Verify that all the appropriate indexes are created over matching rule columns.
2. Verify that the database schema statistics are collected according to the guidelines.

> **See Also:** "Reconciliation Best Practices" on page 14-39 for information about creating indexes for reconciliation and collecting database statistics

Reconciliation Event Does Not Process With Error

The reconciliation event does not process with the following error trace (ORA-31061 error):

```
java.sql.SQLException: ORA-20001: Error occured in XL_SP_ReconBlkChildMthAcntCRUD
while processing Batch ID - 48 XL_SP_ReconBlkChildMthAcntCRUD failed
e_xlSpReconBlkChildMthAcntCRUD - FORMAT_ERROR_BACKTRACE: ORA-06512: at
"SYS.DBMS_XMLGEN", line 7
ORA-06512: at "SYS.DBMS_XMLGEN", line 147
ORA-06512: at "OGD_OIM.OIM_SP_RECONBLKACCOUNTCHGLOG", line 305
ORA-06512: at "OGD_OIM.XL_SP_RECONBLKCHILDMTHACNTCRUD", line 3528

FORMAT_ERROR_STACK: ORA-31061: XDB error: special char to escaped char conversion
failed.
 -31061 -ERROR- ORA-31061: XDB error: special char to escaped char conversion
failed.
ORA-06512: at "OGD_OIM.XL_SP_RECONBLKCHILDMTHACNTCRUD", line 3617
ORA-06512: at "OGD_OIM.XL_SP_RECONBLKACNTRQDCMTCHCRUD", line 91
ORA-06512: at line 1
at oracle.jdbc.driver.T4CTTIoer.processError(T4CTTIoer.java:462)
.
.
.
.
.
.
at weblogic.jdbc.wrapper.PreparedStatement.execute(PreparedStatement.java:99)
at
oracle.iam.reconciliation.dao.ReconActionDao$1ReconDBCall$1.process(ReconActionDao
.java:1489)
at
oracle.iam.reconciliation.dao.ReconActionDao$1ReconDBCall$1.process(ReconActionDao
.java:1472)
at
oracle.iam.platform.tx.OIMTransactionCallback.doInTransaction(OIMTransactionCallba
ck.java:13)
at
oracle.iam.platform.tx.OIMTransactionCallback.doInTransaction(OIMTransactionCallba
ck.java:6)
at
org.springframework.transaction.support.TransactionTemplate.execute(TransactionTem
plate.java:128)
```

This is a database-related issue. For a description of this issue, see technote "XDB reports ORA-31011 if content contains illegal characters (Doc ID 8246403.8)" at My Oracle Support web site at:

https://support.oracle.com

The fix for this enhancement is available in Database version 11.2 onwards, but it is disabled by default. To fix this issue, the enhancement must be enabled.

To workaround this issue, remove the special character in the data.

14.9.3 Troubleshooting Reconciliation Profile Configuration Failures

For any issues related to profile configuration failures, validate the profile by using the ProfileValidator Mbean available in Oracle Enterprise Manager. If the profile is invalid, then the profile is displayed along with the cause of the invalid profile.

Table 14–6 lists the troubleshooting steps that you can perform if you encounter reconciliation errors.

Table 14–6 Troubleshooting Reconciliation Profile Configuration Failures

Problem	Solution
The profile is invalid, and it fails to load with the following exceptions: oracle.iam.reconciliation.exception.ConfigNotFoundException OR oracle.iam.reconciliation.exception.Config with internal exception org.xml.sax.SAXParseException	Perform any one of the following: ■ The exact problem can be diagnosed and fixed by checking the schema validation message. ■ Validate the reconciliation profile XML by using the MBean in Oracle Enterprise Manager. ■ Validate the reconciliation profile by importing the profile and the XSD into an XML schema-aware editor and validate against that schema in that editor, which can point to the exact cause of the failure.
Importing a valid reconciliation profile XML into a system fails to create the necessary configurations.	Check the profile.configure attribute. The value of this attribute must be true. Check the profile.active attribute. The value of this must be true. Or if the attribute is not present, then it means that profile.active is true.
The following error is generated: `oracle.iam.reconciliation.exception.ReconciliationException: Exception occurred while inserting data into table` *`STAGING_TABLE_NAME`* `due to STAGING_TABLE_NAME` This means that a valid reconciliation profile is loaded, but it has not created any configuration in Oracle Identity Manager.	Check the profile.configure and profile.active attributes.

Note: The ProfileValidator Mbean is available for validating the reconciliation profile. This MBean can be accessed by using Oracle Enterprise Manager.

14.9.4 Troubleshooting LDAP Reconciliation Issues

This section describes the following issue related to LDAP reconciliation:

LDAP User Create and Update Reconciliation Scheduled Job Fails With Error

In an integrated environment of Oracle Identity Manager with Oracle Unified Directory (OUD) and libOVD, the LDAP User Create and Update Reconciliation scheduled job fails with the following error:

```
"Invalid syntax of the provided cookie"
```

This error occurs when a numeric change number value, for example 0, is being passed to libOVD when OUD is used as LDAP, and External Change Log (ECL) is enabled in the libOVD adapter. In the scheduled job page, the numeric value, for example 0, is set in the Last Change Number field. This attribute is used to specify the last changelog identifier processed by this scheduled job.

For OUD, when ECL mode is on, the value of the Last Change Number field must not be a numeric value but a string value, such as:

```
"dc=us,dc=oracle,dc=com:00000142b752f5cf473900000ce1;"
```

The following is an example command to get the expected value:

```
ldapsearch -h localhost -p 1389 -D "cn=Directory Manager" -w welcome1 -b "" -s
base "objectclass=*" lastExternalChangelogCookie
dn: lastExternalChangelogCookie:
dc=us,dc=oracle,dc=com:00000142b752f5cf473900000ce1;
```

External Changelog Cookie Expiration Issue When Performing Reconciliation with OUD

For a description of the issue and troubleshooting information, see "Fixing External Changelog Cookie Expiration Issue When Performing Reconciliation with OUD" in the *Oracle Fusion Middleware Integration Guide for Identity Management Suite*.

14.10 Populating Data in the RECON_EXCEPTIONS Table

The RECON_EXCEPTIONS table in Oracle Identity Manager database is used to capture error messages generated during account reconciliation. This data is collected for the purpose of generating reports.

> **See Also:** "Account Reconciliation" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager* for information about account reconciliation

If a reconciliation match is found to a deleted user, then you must insert USER_DELETED in the REX_EXCEPTION column and the key of the deleted user in the USR_KEY column of the RECON_EXCEPTIONS table.

If no match is found, then insert USER_NOT_FOUND in the REX_EXCEPTION column.

If account match is found, then check if the account is already deprovisioned. Then insert into RECON_EXCEPTIONS table with the value `RESOURCE_DEPROVISIONED` in the REX_EXCEPTION column for the user who is to be provisioned.

To populate the RECON_EXEPTIONS table with exception data:

1. Fetch all the events with the change type != ('Modify' , 'Delete') and event status as ('Single User Match Found', 'Single Org Match Found').
2. Provision the resource object for the entities by performing the following:

a. Collect the exception data from RECON_EXCEPTION DB table. To do so, perform any one of the following:

   – Check if the value of the XL.EnableExceptionReports property is TRUE. If it is set to TRUE, then continue to the next step. Otherwise, do not collect the exception data.

   – Select the obj_initial_recon_date in the obj table for the resource object being provisioned, and check if it is earlier than today's date. If an earlier date is displayed, then continue to the next step. Otherwise, do not collect the exception data.

b. While provisioning the resource object to the user, check if the resource object has already been deprovisioned in Oracle Identity Manager:

   – If the resource object is already deprovisioned, then insert into RECON_EXCEPTIONS table the value `RESOURCE_DEPROVISIONED` in the REX_EXCEPTION column for the user who is to be provisioned.

   – If the resource object is not deprovisioned, then insert into RECON_EXCEPTIONS table the value `RESOURCE_NEVER_PROVISIONED` in the REX_EXCEPTION column for the user who is to be provisioned.

14.11 Reconciliation Best Practices

This section describes how to improve performance by identifying indexes that are required for connector tables and reconciliation tables. It contains the following topics:

- Additional Indexes Requirement for Matching Module
- Collecting Database Schema Statistics for Reconciliation Performance

Note: Oracle recommends configuring both the entitlement attribute and the key attribute for the child data in reconciliation field mappings to enable effective duplicate entitlement or child data validation. See "Duplicate Validation for Entitlement or Child Data" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager* for information about duplicate validation for entitlements or child data.

14.11.1 Additional Indexes Requirement for Matching Module

When Oracle Identity Manager is installed, the necessary indexes are created in the Oracle Identity Manager database schema. However, there can be additional indexes required because of dynamic nature of some of the features in Oracle Identity Manager. This is especially true for reconciliation.

Reconciliation uses matching algorithm to find if the user/account/role/organization for which the change is requested is already existing in Oracle Identity Manager or not. The matching algorithm compares the data in set of columns in Oracle Identity Manager with the data in target horizontal table columns. The columns that contains the matching rules are defined in the reconciliation profile. To improve the performance of the matching operation quickly, there must be correct indexes created on the matching rule columns.

To illustrate the recommended method of identifying the appropriate indexes, a sample Active Directory (ADUser) profile present in the Meta Data Store (MDS) repository is taken as an example.

Selecting Indexes for Reconciliation

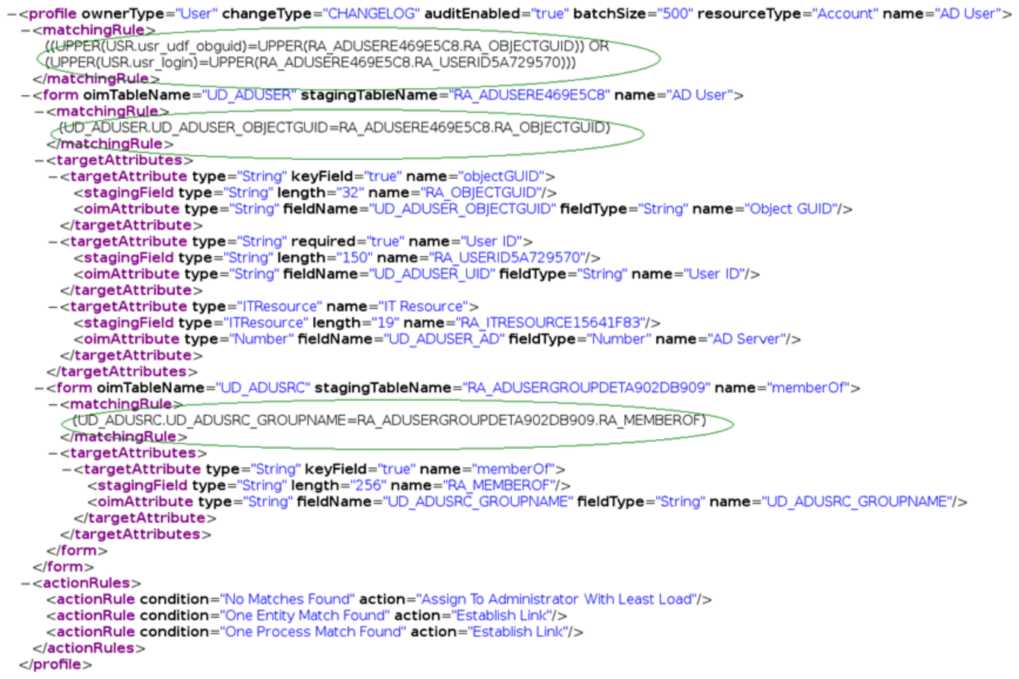
To select indexes based on the matching rule criteria:

1. Open the AD profile file in a text editor.

   Note: The AD user profile must be imported from the MDS by using the Oracle Enterprise Manager, as described in "Migrating User Modifiable Metadata Files" on page 24-1.

2. Search for all occurrences of <matchingRule> tag element in the AD profile, as shown in Figure 14–4:

Figure 14–4 The <matchingRule> Tag Element



```
- <profile ownerType="User" changeType="CHANGELOG" auditEnabled="true" batchSize="500" resourceType="Account" name="AD User">
  - <matchingRule>
      ((UPPER(USR.usr_udf_obguid)=UPPER(RA_ADUSERE469E5C8.RA_OBJECTGUID)) OR
      (UPPER(USR.usr_login)=UPPER(RA_ADUSERE469E5C8.RA_USERID5A729570)))
    </matchingRule>
  - <form oimTableName="UD_ADUSER" stagingTableName="RA_ADUSERE469E5C8" name="AD User">
    - <matchingRule>
        (UD_ADUSER.UD_ADUSER_OBJECTGUID=RA_ADUSERE469E5C8.RA_OBJECTGUID)
      </matchingRule>
    - <targetAttributes>
      - <targetAttribute type="String" keyField="true" name="objectGUID">
          <stagingField type="String" length="32" name="RA_OBJECTGUID"/>
          <oimAttribute type="String" fieldName="UD_ADUSER_OBJECTGUID" fieldType="String" name="Object GUID"/>
        </targetAttribute>
      - <targetAttribute type="String" required="true" name="User ID">
          <stagingField type="String" length="150" name="RA_USERID5A729570"/>
          <oimAttribute type="String" fieldName="UD_ADUSER_UID" fieldType="String" name="User ID"/>
        </targetAttribute>
      - <targetAttribute type="ITResource" name="IT Resource">
          <stagingField type="ITResource" length="19" name="RA_ITRESOURCE15641F83"/>
          <oimAttribute type="Number" fieldName="UD_ADUSER_AD" fieldType="Number" name="AD Server"/>
        </targetAttribute>
      </targetAttributes>
    - <form oimTableName="UD_ADUSRC" stagingTableName="RA_ADUSERGROUPDETA902DB909" name="memberOf">
      - <matchingRule>
          (UD_ADUSRC.UD_ADUSRC_GROUPNAME=RA_ADUSERGROUPDETA902DB909.RA_MEMBEROF)
        </matchingRule>
      - <targetAttributes>
        - <targetAttribute type="String" keyField="true" name="memberOf">
            <stagingField type="String" length="256" name="RA_MEMBEROF"/>
            <oimAttribute type="String" fieldName="UD_ADUSRC_GROUPNAME" fieldType="String" name="UD_ADUSRC_GROUPNAME"/>
          </targetAttribute>
        </targetAttributes>
      </form>
    </form>
  - <actionRules>
      <actionRule condition="No Matches Found" action="Assign To Administrator With Least Load"/>
      <actionRule condition="One Entity Match Found" action="Establish Link"/>
      <actionRule condition="One Process Match Found" action="Establish Link"/>
    </actionRules>
</profile>
```

3. After identifying the columns constituting each matching rule, create the indexes accordingly.

Note:

- If any key field is defined in Oracle Identity Manager as case-insensitive, then a function-based index on that key field must be created. For example, if the connector code internally performs a search for the first name, assuming that FIRST_NAME is a key, then appropriate indexing must be done.
- If multiple or composite keys are used for looking up a user, then choose between individual or composite indexes.
- Pointers for required indexes can also be taken by monitoring the real-time running of reconciliation process from the database side by using a performance-monitoring tool, such as Oracle Enterprise Manager, or through the Automatic Workload Repository (AWR) Reports available in Oracle Database 11*g*.
- To some extent, index creation is automated for profiles created or updated via the Design Console or Deployment Manager import. Validate the automatically created indexes per the rules defined in this section. You must rectify the indexes manually if there are any issues. For profiles created or updated manually, the indexes are not automated and must be created manually. In addition, there is no automation for dropping the indexes if the matching rule field has changed. Dropping indexes must be done manually.
- The list of existing indexes can be viewed on Oracle Enterprise Manager by using the ProfileValidator Mbean.
- Index names starting with RDX are reserved for default reconciliation indexes and must not be used for any custom index creation.

14.11.2 Collecting Database Schema Statistics for Reconciliation Performance

Database statistics is essential for the Oracle optimizer to select an optimal plan in running the SQL queries. Oracle recommends that the statistics are collected regularly for Oracle Identity Manager schema. Because Oracle Identity Manager 11*g* Release 2 (11.1.2.3.0) uses lot of database SQL features for reconciliation process, make sure that the schema statistics are updated before running the reconciliation.

Note:

- Other options with DBMS_STATS.GATHER_SCHEMA_STATS API can be used as required, such as DEGREE,ESTIMATE_PERCENT based on the environment, data profile, Oracle DB Edition and underlying hardware capabilities.
- See "Database Performance Monitoring" in the *Oracle Fusion Middleware Performance and Tuning Guide* for more information about collecting database schema statistics.

Because Oracle Identity Manager reconciliation process is a data-intensive process and quickly brings in large volume of data, database statistics must also be able to represent the underlying data correctly. To achieve this, refer to the following guidelines:

- Make sure that statistics is collected for reconciliation on a fresh setup or with a low volume with no or negligible existing data in the Oracle Identity Manager schema. Maximum row count in relevant Oracle Identity Manager tables must be between 100 and 1000 rows. Examples of tables are USR table for trusted source reconciliation and parent account table for target resource reconciliation.
- For the statistics to be a proper representative of data distribution after reconciliation has started and is expected to bring in a large volume of data, such as more than 20000 users or accounts, collect Oracle Identity Manager schema statistics in the following manner:

  a. Plan to gather statistics after the initial collection only after reconciliation has started successfully and has been running for a while. To verify this, check the counts of a few key tables from the Oracle Identity Manager schema, such as USR table for trusted source reconciliation and parent account (UD_*) tables for target resource reconciliation.

  b. After reconciliation has brought in almost 20000 to 25000 rows in the USR table or in the parent account tables, statistics can be collected.

  Note:

  - Statistics can be gathered concurrently with reconciliation running.
  - The row counts specified in the guidelines are examples and you can determine any other row count for collecting statistics.

- After the statistics is collected, the performance might not improve immediately. However, as older SQL Plans are cleared from the shared pool of the Oracle Database, new and more efficient plans are created and performance improves.

14.12 Monitoring Reconciliation Performance Using DMS

Dynamic Monitoring Service (DMS) commands are used to view performance metrics and configure event tracing. The following DMS matrices are relevant for monitoring reconciliation performance:

- OIM_ScheduledJob: The time taken by a particular scheduled job run.
- Reconciliation Service (ReconOperationsService Or tcReconciliationOpIntf): The time taken by each API on the reconciliation service for creating an event. Connector throughput can be calculated as 'Total Scheduled Job time – Total time for creating the events.
- OIM_JMS: ActionTask shows the consolidated time taken for processing both by the Stored Procedure and the Post Processing done via Orchestration. XLAuditMessage provides information about actual audit processing.
- OIM_EventHandlers: Time taken by each eventhandler within an orchestration.
- DMS Dump: If unable to resolve performance and functional issues, then DMS dumps should be provided for analysis. The command is:

```
${mwhome}/oracle_common/common/bin/wlst>>dms.log
Connect('adminusername','adminpassword');
dumpMetrics(format='xml');
Exit();
```

For detailed information about the command, see the "dumpMetrics" section in "DMS Custom WLST Commands" of the *Oracle Fusion Middleware WebLogic Scripting Tool Command Reference*.

15 Using the Bulk Load Utility

Oracle Identity Manager may be one among many repositories of entity data in your organization. When you start using Oracle Identity Manager, you might want to load data from the other repositories into Oracle Identity Manager. The Bulk Load utility offers a solution to this requirement.

The Bulk Load utility is aimed at automating the process of loading a large amount of data into Oracle Identity Manager. It helps reduce the downtime involved in loading data. You can use this utility after you install Oracle Identity Manager or at any time during the production lifetime of Oracle Identity Manager. The Bulk Load utility can load users, accounts, roles, role hierarchy, role membership, role category data, and organizations.

This document is divided into the following sections:

15.1 Modes of Running the Utility

The Bulk Load utility can be run in one of the following modes:

- **Offline mode:** This is the traditional or existing mode. To run the utility in offline mode, Oracle Identity Manager must be down.

- **Online mode:** In online mode, there is no need to shut down Oracle Identity Manager. online mode only implies that the utility can be run when Oracle Identity Manager is up and running. It is still a command-line utility and no other interface is available for online mode.

By default, Bulk Load utility runs in online mode.

To choose between the online or offline mode, consider the following factors:

- At times, service availability is more important for business reasons. Choose default online mode in this case.
- If volume of new entities loaded is not huge, cost of service restart and index rebuild after offline bulkload is higher than slight performance degradation in online mode. Choose default online mode in this case.
- When load volume is high and existing system data is less in comparison, offline bulk load might have some advantages.

15.2 Features of the Bulk Load Utility

The following are features of the bulk load utility:

- Data can be loaded into Oracle Identity Manager as OIM Users, accounts allocated (provisioned) to OIM Users, roles, role hierarchies, role memberships, role categories, or organizations.
- Data can be loaded from a single or multiple CSV files or a database table. Data imported into Oracle Identity Manager is automatically converted into OIM Users, accounts provisioned to OIM Users, roles, role hierarchies, role memberships, role categories, or organizations.
- Data can be loaded from a single or multiple trusted sources.
- Data can be loaded into either an empty Oracle Identity Manager repository or an Oracle Identity Manager repository that already contains data about OIM Users and resources. In other words, user data can be loaded at any time, either immediately after Oracle Identity Manager installation or when the system is already in production.
- The utility is for creating new entities only. It cannot be used to update or delete existing entities.
- Exceptions generated during user data loading are handled, and records that fail the loading process can be retried.
- Audit snapshots can be generated after a bulk load operation for users.
- After bulk loading of OIM User data, password change at first login is enforced because a dummy password is used during the operation.

Note: You cannot use the utility to encrypt user attributes. In other words, if a user field in Oracle Identity Manager is encrypted, then the utility cannot be used to encrypt data that is loaded into that field.

- The Bulk Load utility can be used in offline or online modes.

15.3 Prerequisites for Running the Bulk Load Utility

Running the Bulk Load utility has the following prerequisites:

- Installing the Bulk Load Utility
- Preparing Your Database for a Bulk Load Operation

15.3.1 Installing the Bulk Load Utility

To install the utility:

1. Zip and copy the following directory from the installation package into a directory on the Oracle Identity Manager database host computer:

   MIDDLEWARE_HOME/Oracle_IDM1/server/db/oim/oracle/Utilities/oimbulkload

   > **Note:** You can run the utility from a remote host. It is not mandatory to run the utility from a directory in the Oracle Identity Manager database host.
   >
   > The utility can also be run directly from the *MIDDLEWARE_HOME*/Oracle_IDM1/server/db/oim/oracle/Utilities/oimbulkload/ directory.

2. Extract the contents of the ZIP file.

   The oimbulkload directory is created when you extract the contents of the ZIP file. The following directories are created inside this directory:

   - sqls: This directory contains SQL scripts used during bulk load operations.
   - scripts: This directory contains the .sh and .bat scripts used during bulk load operations.
   - csv_files: If you are going to use a single or multiple CSV files as the input source, then the CSV files must be stored in this directory.
   - lib: The directory contains the oimBulkLoad.jar file.
   - sample_data: This directory contains the following sample CSV files:
     - For OIM User load operations:

       master.txt

       OIDusers.csv

       HRusers.csv
     - For account load operations:

       parentAD.csv

       childAD.csv
     - For role-related load operations:

       Role.csv (Role load)

       Rolec.csv (Role category)

       Roleh.csv (Role hierarchy)

       Rolem.csv (Role membership)
   - Logs_ *YYYYMMDD_hhmi*: The log directory contains the log files that store the summary of the bulk load operation. This directory is created at run time.

The following sections provide additional information about the utility and bulk load operations:

- Scripts That Constitute the Utility
- Temporary Tables Used During a Bulk Load Operation
- Options Offered by the Utility

15.3.1.1 Scripts That Constitute the Utility

The following are the main scripts that constitute the utility:

- **oim_blkld.bat and oim_blkld.sh**

  This script contains the code to perform bulk load operations. When it is run, this script calls other scripts and stored procedures.

- **oim_blkld_setup.sql**

  This script is used to add a datafile in the Oracle Identity Manager tablespace and provide additional grants to the Oracle Identity Manager database user to perform required operations during Bulk Load. See "Creating a Datafile in the Oracle Identity Manager Tablespace" on page 15-6 for more information.

15.3.1.2 Temporary Tables Used During a Bulk Load Operation

The following temporary tables are used during a bulk load operation:

- **OIM_BLKLD_TMP_*SUFFIX***

  If you are using a CSV file as the input source, then the utility automatically creates the OIM_BLKLD_TMP_*SUFFIX* table and first loads data from the CSV file into this table. The suffix for the table name is determined as follows:

  - The first 6 characters of the file name are taken into account.
  - Special characters in the file name and the file extension (.csv) are ignored while determining the first 6 characters.
  - A unique number is appended to the first 6 characters.
  - For example, if the name of the file is acc_Data.csv, then the table that is created during the bulk load operation is named `oim_blkld_tmp_accDat1`.

  If there are multiple CSV files, then one table is created for each file. Because the first six characters of each CSV file name are appended to the table name, you must ensure that the first six characters of each file's name are unique. This guideline is explained later in this document.

  Note: if you are using a database table as the input source, then you can specify any name for the table. You provide the name of this table as one of the input parameters of the utility.

- **OIM_BLKLD_EX_*SUFFIX***

  The OIM_BLKLD_EX_*SUFFIX* table is used to hold data records that fail (are not loaded into Oracle Identity Manager) during a bulk load operation. One OIM_BLKLD_EX_SUFFIX table is created for each OIM_BLKLD_TMP_*SUFFIX* table. The EXCEPTION_MSG column of the table stores the reason for failure of each record in the table.

If you are using CSV files as the input source, then the first six characters of the CSV file name are added as a suffix to the table name. For example, if the name of the CSV file is usrdt120508.csv, then the name of the table is OIM_BLKLD_EX_usrdt1. If there are multiple CSV files, then one temporary table is created for each CSV file.

> **Note:** If there are multiple CSV files, then you must ensure that the first six characters of each CSV file name are unique.

- **OIM_BLKLD_LOG**

  During a bulk load operation, the utility inserts progress and error messages in the OIM_BLKLD_LOG table. You can query this table to monitor the progress of a bulk load operation. This procedure is described in detail later in this document.

15.3.1.3 Options Offered by the Utility

When you run the bulk load utility, it prompts you to select one of the following options:

> **Note:** The utility prompts for more input depending on the option you select.

- Load User Data

  You select this option if you want the utility to load OIM User data. In other words, data is imported into the USR table of Oracle Identity Manager. You can select the input source, CSV files or database tables, for the data that you want to load.

- Load Account Data

  You select this option if you want the utility to load account data. In other words, data is imported into the relevant UD_ tables of Oracle Identity Manager. You can select the input source, CSV files or database tables, for the data that you want to load.

- Load Role Data

  You select this option if you want the utility to load role data. In other words, data is imported into the UGP table of Oracle Identity Manager. You can select the input source, CSV files, or database tables, for the data that you want to load.

- Load Role Membership

  You select this option if you want the utility to load role membership data. In other words, data is imported into the USG table of Oracle Identity Manager. You can select the input source, CSV files or database tables, for the data that you want to load.

- Load Role Hierarchy

  You select this option if you want the utility to load role hierarchy data. In other words, data is imported into the GPG table of Oracle Identity Manager. You can select the input source, CSV files, or database tables, for the data that you want to load.

- Load Role Category

You select this option if you want the utility to load role data. In other words, data is imported into the ROLE_CATEGORY tables of Oracle Identity Manager. You can select the input source, CSV files, or database tables, for the data that you want to load.

- Generate Audit Snapshot

  You select this option if you want the utility to generate an audit snapshot of users that you have loaded.

15.3.2 Preparing Your Database for a Bulk Load Operation

Preparing your database for a bulk load operation involves the following:

- Creating a Tablespace for Temporary Tables
- Creating a Datafile in the Oracle Identity Manager Tablespace

15.3.2.1 Creating a Tablespace for Temporary Tables

As mentioned in "Temporary Tables Used During a Bulk Load Operation", temporary database tables are used during the bulk load operation. It is recommended that you create a tablespace to accommodate these temporary tables instead of using the default tablespace of the Oracle Identity Manager database.

Follow the instructions in the database documentation to create a tablespace.

15.3.2.2 Creating a Datafile in the Oracle Identity Manager Tablespace

The default size of the datafile in the Oracle Identity Manager tablespace created during Oracle Identity Manager installation is 500 MB. You may need to add space to this datafile to accommodate the data that you are going to load. The alternative is to create a datafile.

To create a datafile in the Oracle Identity Manager tablespace:

1. Start a SQL*Plus session.
2. Connect to the Oracle Identity Manager database as SYSDBA.
3. Run the oim_blkld_setup.sql script. The script will prompt for the following:
   - Name of the Oracle Identity Manager tablespace
   - Full path and name for the datafile to be added in the Oracle Identity Manager tablespace
   - Oracle Identity Manager database user name

After providing input to prompted Oracle Identity Manager database user name, appropriate grants to perform required operations during Bulk Load are provided to the database user.

15.4 Running the Utility

Note: If there are name conflicts with existing tables, then the utility overwrites existing temporary tables at the start of each run. If required, rename temporary database tables created during an earlier run of the utility.

To run the utility:

Note: If the underlying database version is 12*c* (12.x), then before starting with the utility execution steps described in this section, perform **any one** of the following steps:

- Copy the `ojdbc5.jar` file from RDBMS 11*g* binaries directory location $*ORACLE_HOME*/jdbc/lib/ to RDBMS 12c binaries directory location $*OIM_HOME*/server/db/oracle/Utilities/oimbulkload/lib/.
- Download the `ojdbc5.jar` file from the Oracle web site (Oracle Database 11g Release 2 JDBC Drivers) at the following URL:

  `http://www.oracle.com/technetwork/apps-tech/jdbc-112010-090769.html`

  Copy the `ojdbc5.jar` file to RDBMS 12*c* binaries directory location $*OIM_HOME*/server/db/oracle/Utilities/oimbulkload/lib/.

---

1. To run the utility in offline mode, stop Oracle Identity Manager. To run in online mode, there is no need to shut down Oracle Identity Manager.
2. Run one of the following scripts:

   Note: To load CSV file with non-ASCII data, before running the oim_blkld.sh or oim_blkld.bat script, set the NLS_LANG environment parameter to the UTF8 characterset, in the following format:

   NLS_LANG = *LANGUAGE_TERRITORY*.UTF8

   For example:

   NLS_LANG = American_America.UTF8

   - On UNIX computers:
     - To run in online mode, run:

       ```
       OIMBulkload/script/oim_blkld.sh
       ```

       OR

       ```
       OIMBulkload/script/oim_blkld.sh -online
       ```

     - To run in offline mode, run:

       ```
       OIMBulkload/script/oim_blkld.sh -offline
       ```

   - On Microsoft Windows computers:
     - To run in online mode, run:

       ```
       OIMBulkload\script\oim_blkld.bat
       ```

       OR

       ```
       OIMBulkload\script\oim_blkld.bat -online
       ```

     - To run in offline mode, run:

```
OIMBulkload\script\oim_blkld.bat -offline
```

Note: OIMBlukload is the directory in which the scripts/sqls/csv_files/lib/sample_data directories are present.

---

3. From the main menu, select one of the options depending on the data you want to load, such as user, account, or role-related data, as described in "Options Offered by the Utility" on page 15-5.
4. From the second menu:
   - Select **CSV File** if you are using CSV files as the input source.
   - Select **DB Table** if you are using a database table as the input source.
5. When prompted, provide values for the input parameters described in "Determining Values for the Input Parameters of the Utility" on page 15-14.

Note: See "Determining Values for the Input Parameters of the Utility" on page 15-14 for information about the input parameters required for loading OIM User data. See corresponding sections for information about the input parameters required to load account, role, role hierarchy, role membership, and role category data.

---

6. Monitor the performance of the operation by following the steps given in "Monitoring the Progress of the Operation".

15.5 Performance Best Practices for Bulk Load

To enhance the performance of the Account Bulk Load operation:

1. Split the data load in phases for a high-volume entity data load for Users/Accounts/Role Membership, for example, when data load is greater than 1 million for Users and greater than 250 thousand for Accounts.
2. The phase-wise load can be in the initial size of 500 thousand, and thereafter, in the size of 2 to 3 million Entity data.
3. Perform Stats gathering operation essentially after the first and second batch of data load.

   For information about Stats gathering operation, see "Monitoring Oracle Identity Manager Performance" in the *Oracle Fusion Middleware Performance and Tuning Guide*.
4. For Account data load, when the source is database table, then make sure that relevant indexes are present on the columns as per the reconciliation matching rules.

   For further details please refer to the documentation on "Additional Indexes Requirement for Matching Module" in the *Oracle Fusion Middleware Developer's Guide for Oracle Identity Manager*.
5. Oracle recommends loading organization data in online mode.

15.6 Loading OIM User Data

The following is a summary of the steps involved in loading OIM User data:

1. Prepare your database for bulk load if not done already. See "Preparing Your Database for a Bulk Load Operation" on page 15-6 for details.
2. Create the OIM User whose password will be used as the default password for all OIM Users created during the bulk load operation.
3. Create the input source for the bulk load operation.

   If you want to use a database table as the input source, then create the table and copy user data into the table.

   If you want to use CSV files as the input source, then create the CSV files and copy user data into the files. In addition, create a master.txt file containing the names of the files in the sequence in which you want to load data from them.
4. Determine values for the input parameters of the utility.
5. Stop Oracle Identity Manager if you want to run in offline mode. For online mode, Oracle Identity Manager server can be running.
6. Run the oim_blkld.sh or oim_blkld.bat script. See "Running the Utility" on page 15-6 for information about running the oim_blkld.sh or oim_blkld.bat scripts.
7. Monitor the progress of the bulk load operation.
8. Determine the outcome of the bulk load operation.
9. If required, reload data that was not loaded during the first run.
10. Restart Oracle Identity Manager if it was stopped in step 5.
11. Verify the outcome of the bulk load operation.
12. Gather diagnostic data from the operation.
13. Remove temporary tables and files created during the operation.
14. Generate an audit snapshot.

The following sections provide detailed information about the steps involved in loading OIM User data:

- Setting a Default Password for OIM Users Added by the Utility
- Creating the Input Source for the Bulk Load Operation
- Determining Values for the Input Parameters of the Utility
- Monitoring the Progress of the Operation
- Handling Exceptions Recorded During the Operation
- Fixing Exceptions and Reloading Data Records
- Verifying the Outcome of the Bulk Load Operation
- Generating an Audit Snapshot

15.6.1 Setting a Default Password for OIM Users Added by the Utility

The utility does not encrypt passwords that it assigns to OIM Users created during the bulk load operation. Instead, it assigns the password of an existing OIM User to all OIM Users that are created during the operation.

Note: Each OIM User is required to change the password at first login.

When you run the utility, it prompts for the login name of the existing OIM User whose password you want to use as the default password for the new OIM Users. Before you run the utility, create this OIM User as follows:

Note: You can create a user in Oracle Identity Manager dedicated for the bulk load operation, and later delete the user if it not required any more. Otherwise, any existing OIM User can be used to perform bulk load operations.

1. Log in to the Oracle Identity Self Service as a user with Create User privileges.
2. On the left navigation pane, under Administration, click **Users**. The Search Users page is displayed.
3. From the Actions menu, select **Create**. The Create User page is displayed with input fields for user profile attributes.
4. Specify values for the following fields:
   - User Login
   - First Name (optional)
   - Last Name
   - Organization: Select **Xellerate Users**.
   - Password
   - Confirm Password
5. Click **Submit**.

15.6.2 Creating the Input Source for the Bulk Load Operation

Depending on the input source that you want to use, apply the guidelines given in one of the following sections:

- Using CSV Files As the Input Source
- Creating Database Tables As the Input Source

15.6.2.1 Using CSV Files As the Input Source

If you want to use CSV files as the input source for the bulk load operation, then apply the following guidelines while creating the CSV files:

- The CSV files must be placed in the oimbulkload/csv_files directory.
- The first line in the CSV file is called the control line. This line must contain a comma-separated list of column names of the USR table in the Oracle Identity Manager database.

Note: Ensure that the Password column or any other encrypted column is not included in the list of columns. As mentioned earlier in this document, the utility assigns the password of an existing OIM User that you specify to all OIM Users that it loads into Oracle Identity Manager.

- From the second line onward, the file must contain values for the columns in the control line. The order of columns in the first line and the values in the rest of the lines must be the same.

  The following are sample contents of a CSV file:

  ```
  USR_LOGIN,USR_FIRST_NAME,USR_LAST_NAME,UD_ADUSER_OBJECTGUID
  john_doe, John, Doe, jdoe
  jane_doe, Jane, Doe, janedoe
  richard_roe, Richard, Roe, rroe
  ```

- If the value in any column contains a comma, then that value must be enclosed in double quotation marks (").
- The CSV file must contain values for all columns that are designated as mandatory in the USR table. The following table lists the mandatory columns required to load the USR table:

Mandatory Column	Description
USR_FIRST_NAME	The first name of the user
USR_LAST_NAME	The last name of the user

Note:

- USR_LOGIN is not a mandatory column in Oracle Identity Manager 11*g* Release 2 (11.1.2.3.0).
- There are some key mandatory columns that you can ignore. For example, the ACT_KEY column in the USR table, which is populated by ORG_NAME.

- Each row in the CSV file must have a unique value for the USR_LOGIN column in the USR table. If there are multiple files, you must ensure that USR_LOGIN values are unique across the CSV files. This check for uniqueness of USR_LOGIN values must also cover existing OIM Users in Oracle Identity Manager.

  Ensuring that USR_LOGIN values are unique can be a time-consuming exercise. As an alternative, you can first perform the bulk load operation, fix USR_LOGIN values that are not unique, and then retry the loading operation for the modified user records. This is possible because the utility checks for uniqueness of USR_LOGIN values at run time and copies records that fail this check into the OIM_BLKLD_EX table. Later in this document, there are instructions on retrying the bulk load operation for records that are not loaded during the first run.
- If you want to include an organization name in each user record, then add ORG_NAME in the control line and enter the organization name for each user from the second line onward. If ORG_NAME is not included, then the users must be assigned to the Xellerate Users organization.

Note: All organization names listed under the ORG_NAME column in the CSV file must exist in Oracle Identity Manager.

- If you want to include a manager name in each user record, then add MANAGER_NAME in the control line and enter the USR_LOGIN value of the manager for each user from the second line onward.

  The utility looks up the USR_LOGIN values for managers after all user data, from all CSV files, is loaded into Oracle Identity Manager. If a USR_LOGIN value given in the MANAGER_NAME column does not exist in Oracle Identity Manager, then the lookup for that user record fails and the record is copied into the exception table, OIM_BLKLD_EX. At the end of the bulk load operation, you can perform the procedure described in "Fixing Exceptions and Reloading Data Records" to reload user records that fail the first run.

- If you want to include a password in each user record, then add *USR_PASSWORD* in the control line, and enter the encrypted password for each user record from the second line onward. The password should be encrypted. A non-encrypted password will be loaded in the system, but user will not be able to login because Bulk Load utility does not encrypt it. Also, the value for *USR_PASSWORD* cannot be blank or NULL.

- Note that the following default values are inserted into Oracle Identity Manager if the CSV file does not contain values for these columns:

  ORG_NAME: `Xellerate Users`

  USR_TYPE: `End-User`

  USR_STATUS: `Active`

  USR_EMP_TYPE: `Full-Time`

- Create a master TXT file containing the names of the CSV files containing user data to be loaded. You can specify any name for the file, for example, master.txt. Save the master file in the oimbulkload/csv_files directory.

  If you want to load multiple CSV files, then enter the name of each data CSV file on a separate line in the master file. Order the list of CSV file names in the sequence in which you want the utility to load data from the files. For example, suppose you have created three data CSV files, London_Users.csv, NewYork_Users.csv, and Tokyo_Users.csv. In the master file, you enter the names of the data CSV files in the following order:

  ```
  Tokyo_Users.csv
  London_Users.csv
  NewYork_Users.csv
  ```

  When you run the utility, data is loaded in this order. This is because the user data in London and New York may have a dependency on the Tokyo users. This is to ensure the manager-user hierarchy.

- If the CSV file is generated on Microsoft Windows and is to be loaded on Linux environment, then remove the special characters, such as '\n\r', to avoid run-time errors.

Note: While copying a CSV file from Windows to UNIX, Solaris, or Linux systems, some special characters, such as `^M`, are appended to the file. This is because, the file from Windows is in DOS (ASCII) format and must be converted to ISO format.

Solaris preinstalls the `dos2unix` utility into the system to do this job. But for UNIX/Linux systems, the CSV file must be converted from DOS format to UNIX format to ensure sanity of the input file before being used in the Bulk Load operation. To do this, the syntax is:

```
# dos2unix CSV_FILE_NAME
```

If the `dos2unix` utility does not exist in the UNIX/Linux systems, then the administrator can install the utility for the respective UNIX/Linux versions by using the relevant documentation.

15.6.2.2 Creating Database Tables As the Input Source

If you want to use a database table as the input source for loading OIM User data, then apply the following guidelines while creating the database table:

- Create the table in the Oracle Identity Manager database.
- The table must contain the following primary key column:

  OIM_BLKLD_USRSEQ NUMBER(19)

  The utility uses this column as the primary key. If required, you can use a database sequence to populate this column.
- The rest of the columns must be the same as the ones in the USR table that you want to use. In other words, ignore optional USR_ columns that you do not want to include in the table that you create.
- Note that the following default values are inserted into Oracle Identity Manager if the table does not contain values for these columns:

  ORG_NAME: `Xellerate Users`

  USR_TYPE: `End-User`

  USR_STATUS: `Active`

  USR_EMP_TYPE: `Full-Time`
- If you want to include an organization name in each user record, then add ORG_NAME in the control line and enter the organization name for each user from the second line onward. If ORG_NAME is not included, then the users must be assigned to the Xellerate Users organization.
- If you want to include a manager name in each user record, then add MANAGER_NAME in the control line and enter the USR_LOGIN value of the manager for each user from the second line onward.
- If you want to include a password in each user record, then add USR_PASSWORD column in the table, and provide the encrypted password for each user record. The password should be encrypted. A non-encrypted password will be loaded in the system, but user will not be able to login because Bulk Load Utility does not encrypt it. Also, the value for *USR_PASSWORD* cannot be blank or NULL.

Table 15–1 shows the structure of a sample database table.

Table 15–1 Structure of a Sample Database Table

Name	Null?	Type
USR_LOGIN	NOT NULL	VARCHAR2(256)
USR_FIRST_NAME		VARCHAR2(150)
USR_LAST_NAME	NOT NULL	VARCHAR2(150)
. . .	. . .	. . .
OIM_BLKLD_USRSEQ	NOT NULL	NUMBER(19)

15.6.3 Determining Values for the Input Parameters of the Utility

The following are input parameters of the utility:

- Oracle Home

  Value of the ORACLE_HOME environment variable on the host computer for the Oracle Identity Manager database

- Database Connection String

  Connection string to connect to the database that must be entered in the following format:

  //HOST_IP_ADDRESS:PORT_NUMBER/SERVICE_NAME

- OIM DB User

  Database login ID of the Oracle Identity Manager database user

- OIM DB Pwd

  Password of the Oracle Identity Manager database user

  The database user password is to be entered twice when prompted.

- Master file name

  Name of the file containing names of the CSV data files to be loaded

  This parameter is used only if the input source is a single or multiple CSV files. You place the master file and CSV data files in the oimbulkload/csv_files directory. See "Using CSV Files As the Input Source" for more information.

- Tmp table name

  Name of the temporary table to be used as the input source

  This parameter is used only if the input source for the bulk load operation is a database table. See "Creating Database Tables As the Input Source" for more information.

- Control Line

  Comma-separated list of names of columns to be loaded from the database table into Oracle Identity Manager

  This parameter is used only if the input source for the bulk load operation is a database table.

- Tablespace Name

  Name of the tablespace in which temporary tables are to be created during the bulk load operation. If the user does not provide the tablespace name, then it will pick the default tablespace.

See "Preparing Your Database for a Bulk Load Operation" on page 15-6 for more information.

- Date format

  Date format used by date columns in the CSV files

  This parameter is used only if the input source is a single or multiple CSV files.

  The date format must match the following:

  – Oracle supported date formats, such as dd-mm-yyyy or MM-DD-YYYY

  – The date format specified in the CSV file

- Batch Size

  Number of user records that must be processed by the utility as a single transaction

  The batch size can influence the performance of the bulk load operation. The default value of this parameter is 10000.

- Debug Flag

  You can specify Y or N as the value of this parameter. If this parameter is set to Y, then the utility records detailed information about events that occur during the bulk load operation. See "Data Recorded During the Operation" on page 15-42 for more information.

- User ID for default password

  Login name of the OIM User that you create by performing the procedure described in "Setting a Default Password for OIM Users Added by the Utility" on page 15-9.

15.6.4 Monitoring the Progress of the Operation

During the bulk load operation, you can query the OIM_BLKLD_LOG table for information about the progress of the operation. For example, you can run the following query to see progress messages generated during the bulk load operation to load OIM User data:

```
SELECT MSG FROM OIM_BLKLD_LOG
WHERE MODULE = 'USER' AND LOG_LEVEL = 'PROGRESS_MSG'
ORDER BY MSG_SEQ_NO;
```

Errors encountered during the bulk load operation can be viewed by querying the OIM_BLKLD_LOG table. The following is an example of the query to retrieve error messages:

```
SELECT MSG FROM OIM_BLKLD_LOG
WHERE MODULE = 'USER' AND LOG_LEVEL = 'ERROR'
ORDER BY MSG_SEQ_NO;
```

15.6.5 Handling Exceptions Recorded During the Operation

At the end of a bulk load operation, the utility records statistics related to the operation in the following file:

oimbulkload/logs_*YYYYMMDD_hhmm*/oim_blkld_user_load_summary.log

To determine if there were exceptions during the operation, open this log file and look for the number against the Number of Records Rejected label. If the number of rejected records is greater than zero, then exceptions were thrown during the operation. User records that are rejected by the utility are recorded in the exception table (OIM_BLKLD_EX_*SUFFIX*). For each rejected record, the EXCEPTION_MSG column in the OIM_BLKLD_EX_*SUFFIX* table stores information about the reason the record could not be loaded.

Example 15–1 shows sample statistics recorded in the log file at the end of a bulk load operation to store OIM User data.

Example 15–1 Sample Log File Generated After Loading OIM User Data

```
************************************************************

Processing File: u10.csv
================================================================
U S E R   L O A D   S T A T I S T I C S   F O R   F I L E : u10.csv
================================================================
Start Time:   08-AUG-08 11.44.12.228000 AM
End Time:     08-AUG-08 11.44.13.368000 AM
Number of Records Processed:  10
Number of Records Loaded:     8
Number of Records Rejected:   2
================================================================
The name of the TMP table used during the load:
OIM_BLKLD_TMP_U101

The name of the Exception table used during the load:
OIM_BLKLD_EX_U101

****************************************************************
Processing File: u10b.csv

================================================================
U S E R   L O A D   S T A T I S T I C S   F O R   F I L E : u10b.csv
================================================================
Start Time:   08-AUG-08 11.44.15.368000 AM
End Time:     08-AUG-08 11.44.15.540000 AM
Number of Records Processed:  16
Number of Records Loaded:     15
Number of Records Rejected:   1
================================================================
The name of the TMP table used during the load:
OIM_BLKLD_TMP_U10B2

The name of the Exception table used during the load:
OIM_BLKLD_EX_U10B2
================================================================

================================================================
Time taken in re-building indexes and enabling FK constraints
================================================================
Start time:      08-AUG-08 11.44.15.556000 AM
End Time:        08-AUG-08 11.46.50.586000 AM
================================================================
```

In this sample, the number of rejected records is 2. If the log file shows that any records were rejected by the utility, then see "Fixing Exceptions and Reloading Data Records" for information about retrying the load operation for these records.

Note: At the end of each bulk load operation, it is recommended that you create a backup of the exception tables.

15.6.6 Fixing Exceptions and Reloading Data Records

As mentioned earlier, errors encountered during the bulk load operation can be viewed by querying the OIM_BLKLD_LOG table. The following is an example of the query to retrieve error messages:

```
SELECT MSG FROM OIM_BLKLD_LOG
WHERE MODULE = 'USER' AND LOG_LEVEL = 'ERROR'
ORDER BY MSG_SEQ_NO;
```

An exception table OIM_BLKLD_EX_*SUFFIX* is created for each data table used as the input source during the bulk load operation. Records that do not meet the criteria for the operation are copied into this exception table. The suffix appended to the name of each exception table is the same as suffix appended to the name of the corresponding data table.

To reload rejected records:

1. Create a backup of the exception table in which rejected records are stored.

   Note: Although this is an optional step, it is recommended that you create a backup.

2. Review each record in the exception table, and fix errors in the data based on the message recorded in the EXCEPTION_MSG column.
3. After you fix errors in all the rejected records in an exception table, rename the table to OIM_BLKLD_TMP_*SUFFIX* and then use it as the input source.
4. Load records from the OIM_BLKLD_TMP_*SUFFIX* table by running the utility. See "Running the Utility" for more information.
5. Repeat Steps 1 through 4 until the Number of Records Rejected label in the oim_blkld_user_load_summary.log file shows the value 0.
6. Restart Oracle Identity Manager if loading was done in offline mode.

Note: Being a database-intensive operation by design, Bulk Load disables the constraints and indexes on the relevant Oracle Identity Manager entity tables during the start of the operation. Bulk Load operation failure towards the end of the load might at times render the indexes and constraints in disabled state. To identify and fix this issue, manually restore the indexes and constraints as follows:

1. Identify the unusable indexes and disabled constraints. To do so, the following SQL queries or similar mechanism can be used:

   ```
   SELECT TABLE_NAME, CONSTRAINT_NAME FROM user_constraints WHERE
   status = 'DISABLED';
   SELECT index_name FROM user_indexes WHERE status = 'UNUSABLE';
   ```

2. Enable the constraints and rebuild the indexes manually, as shown:

   ```
   ALTER TABLE TABLE_NAME ENABLE CONSTRAINT CONSTRAINT_NAME;
   ALTER INDEX INDEX_NAME REBUILD;
   ```

15.6.7 Verifying the Outcome of the Bulk Load Operation

To verify the outcome of the bulk load operation, check if you are able to perform the following steps for one of the OIM User added by the utility:

Note:

- These steps leave footprints in the system, and therefore, the bulk load verification must be performed by using a test user. If you do not want to leave the footprints in the system, then revert the changes. For example, if you have provisioned a resource to a OIM User, then deprovision the resource after testing the outcome of the bulk load operation.
- If Oracle Identity Manager is synchronized with LDAP, then after running the user data upload, run the Bulk Load Post Process scheduled job with the LdapSync option set to `Yes`. If you want to generate the random password and send an email to the user, then you must configure the email notification and set the Generate Password and Notification parameters to Yes in the Scheduler. See "Predefined Scheduled Tasks" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager* for information about the Bulk Load Post Process scheduled job.

- Log in as the OIM User. The system should prompt you to change the password.
- Provision a resource for the OIM User.
- Add the OIM User to a role.
- Modify the account profile of the OIM User.
- Revoked the resource provisioned to the OIM User.
- Unassign the OIM User from the role to which the user was added earlier.
- Modify the account profile again to restore the profile to its original state.
- Check if the User Resource Access report (an operational report) and the User Resource Access History report can be generated for the user.

15.6.8 Generating an Audit Snapshot

If required, you can generate an audit snapshot of Oracle Identity Manager data after a bulk load operation, or at any time during the bulk load operation. You can also generate audit snapshots by selecting option 7 in the Bulk Load utility. The utility uses the audit engine shipped with Oracle Identity Manager. Internally, the GenerateSnapshot script is called when you run the audit utility. Similarly, the GenerateSnapshot script is called when you select the option to generate an audit snapshot.

Note: Oracle Identity Manager must be up and running when you run the audit utility.

Before you generate an audit snapshot, for running the GenerateSnapshot script, you must set the following environment variables:

- APP_SERVER: weblogic
- OIM_ORACLE_HOME: c:\work1\Oracle_IDM1
- JAVA_HOME: C:\jdk160
- MW_HOME: c:\work1
- WL_HOME: c:\work1\wlserver_10.3
- DOMAIN_HOME: C:\work1\user_projects\domains\base_domain

Note: C:\work1\ is a sample directory path of *MW_HOME*.

See "Configuring Auditing" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager* for information about the procedure to generate audit snapshots.

15.7 Loading Account Data

The following is a summary of the steps involved in loading account data:

1. Prepare your database for a bulk load operation, if not already done. See "Preparing Your Database for a Bulk Load Operation" on page 15-6 for details.
2. Create the input source for the bulk load operation.

   If you want to use a database table as the input source, then create the table and copy account data into the table.

   If you want to use CSV files as the input source, then create the CSV files and copy account data into the files.
3. Determine values for the input parameters of the utility.
4. Stop Oracle Identity Manager if you want to run in offline mode. For online mode Oracle Identity Manager server can be running.
5. Run the oim_blkld.sh or oim_blkld.bat script.
6. Monitor the progress of the bulk load operation.
7. Determine the outcome of the bulk load operation.
8. If required, reload data that was not loaded during the first run.

9. Restart Oracle Identity Manager if it was stopped in step 4.
10. Verify the outcome of the bulk load operation.
11. Gather diagnostic data from the operation.
12. Remove temporary tables and files created during the operation.

Requirements and Features of the Bulk Load Operation for Account Data

The following are requirements and features of the bulk load operation for account data:

- Reconciliation must be set up and you should be able to test reconciliation by importing a few accounts from the target system.
- Only accounts for which there are corresponding OIM Users can be loaded.
- A target system that requires multiple IT resources is not supported.
- Duplicate accounts cannot be detected during a bulk load operation. If there are multiple entries for the same account in the input source, then multiple accounts are created for the corresponding OIM User.
- For a particular target system, if there are multiple provisioning processes/process forms in Oracle Identity Manager, then the utility uses the default provisioning process for the resource object.
- Information about the stage up to which earlier bulk load operations progressed is not stored. In other words, the utility cannot resume a bulk load operation. You must backup the Oracle Identity Manager database before a bulk load operation. If you want to retry a bulk load operation, you must first restore the database and then rerun the procedure.
- Bulk Load utility takes the corresponding application instance name as input to load account data. If the application instance name is not known to the user, then Bulk Load utility prompts for the resource object name and IT resource name, based on which account data is loaded.
- Loading accounts where the target system is Active Directory, make sure that the input source (CSV file or database table) have the following attributes as mandatory in the attribute list along with its values:
  - UD_ADUSER_COMMONNAME
  - UD_ADUSER_USERPRINCIPALNAME

  Failing to load values for these attributes at the time of Bulkload can result into failures in the provisioning-related operations at a later stage for this target.
- If you are loading account data with entitlements, then:
  1. Ensure that the lookup recon task has been run to populate the appropriate lookup table, and the table is recent.
  2. Run the Entitlement List scheduled job and let it complete before loading the data. To verify, ensure that all the entitlements are showing up in the ENT_LIST table. See "Predefined Scheduled Tasks" in *Administering Oracle Identity Manager* for information about the Entitlement List scheduled job.

  Failure to do these steps before account load may lead to missing entitlements on user pages. Such situation can be corrected by running the Entitlement Assignments scheduled job to completion after bulk load, which is avoidable if steps 1 and 2 are followed. See "Predefined Scheduled Tasks" in *Administering*

Oracle Identity Manager for information about the Entitlement Assignments scheduled job.

The following sections provide detailed information about the steps involved in loading account data:

- Creating the Input Source for the Bulk Load Operation
- Determining Values for the Input Parameters of the Utility
- Monitoring the Progress of the Operation
- Handling Exceptions Recorded During the Operation
- Fixing Exceptions and Reloading Data Records
- Verifying the Outcome of the Bulk Load Operation

15.7.1 Creating the Input Source for the Bulk Load Operation

Depending on the input source that you want to use, apply the guidelines given in one of the following sections:

- Using CSV Files As the Input Source
- Creating Database Tables As the Input Source

15.7.1.1 Using CSV Files As the Input Source

If you want to use CSV files as the input source for the bulk load operation, then apply the following guidelines while creating the CSV files:

- The CSV files must be placed in the oimbulkload/csv_files directory.
- The first line in the CSV file is called the control line. This line must contain a comma-separated list of column names in the account (UD_*) table into which you want to load the account data. To find out the UD_ table, go to the process form in the Design Console. See Chapter 2, "Developing Provisioning Processes" for information about process forms.

  Note: Ensure that the Password column or any other encrypted column is not included in the list of columns.

- From the second line onward, the file must contain values for the columns in the control line. The order of columns in the first line and the values in the rest of the lines must be the same.
- If the value in any column contains a comma, then that value must be enclosed in double quotation marks (").
- The CSV file must contain values for all columns that are designated as mandatory in the account table. The key mandatory columns in the account table must be ignored.
- If you want to load account data into parent and child tables, then you must create one parent CSV file and one child CSV file for each child table. For example if you are loading data into one parent table and three child tables, then you must create one parent CSV file and three child CSV files.
- If you want to load account data into parent and child tables, then at least one column must be the same in both tables. This column corresponds to the link

attribute between the parent and child CSV files. The following example illustrates this:

The following are sample contents of a parent CSV file:

```
UD_ADUSER_UID,,UD_ADUSER_FNAME,UD_ADUSER_LNAME,UD_ADUSER_MNAME,UD_ADUSER_FULLNA
ME,UD_ADUSER_OBJECTGUID
ADTEST1,"7~CN=ForeignSecurityPrincipals,DC=vivek01,DC=com",adtest1,adtest1,,adt
est1,102
```

The following are sample contents of a child CSV file:

```
UD_ADUSER_UID,UD_ADUSRC_GROUPNAME
ADTEST1,"7~CN=ForeignSecurityPrincipals,DC=vivek01,DC=com",group2
```

The UD_ADUSER_UID column is common to both the parent file and the child file.

Note:

- The UD_ADUSER_OBJECTGUID column is mandatory in the parent CSV file for loading accounts by using the bulk load operation. This column must be added to the parent CSV file in spite of nullable column in the database.
- Common key column being defined in parent and child CSV need not be present in the child database table. This column is being used by the BulkUpload utility to identify the record key and make the respective entries in corresponding child tables. For example, UD_ADUSER_UID.

  This column should be present in parent table but need not be present in any of the child table.

- If the CSV file is generated on Microsoft Windows and is to be loaded on Linux environment, then remove the special characters, such as '\n\r', to avoid run-time errors.

15.7.1.2 Creating Database Tables As the Input Source

If you want to use a database table as the input source for loading account data, then apply the following guidelines while creating the database table:

- Create the table in the Oracle Identity Manager database.
- The table must contain the following primary key column:

  OIM_BLKLD_USRSEQ   NUMBER(19)

  The utility uses this column as the primary key. If required, you can use a database sequence to populate this column.

- The rest of the columns must be the same as the ones in the account (UD_) table that you want to use. In other words, ignore optional UD_ columns that you do not want to include in the table that you create.

Table 15–2 shows the structure of a sample parent table.

Table 15–2 Structure of a Sample Database Table

Name	Null?	Type
UD_ADUSER_UID		VARCHAR2(20)
UD_ADUSER_ORGNAME		VARCHAR2(256)
UD_ADUSER_FNAME		VARCHAR2(80)
UD_ADUSER_LNAME		VARCHAR2(80)
UD_ADUSER_MNAME		VARCHAR2(80)
UD_ADUSER_FULLNAME		VARCHAR2(240)
OIM_BLKLD_SEQ	NOT NULL	NUMBER(19)

Table 15–3 shows the structure of a sample child table.

Table 15–3 Structure of a Sample Child Database Table

Name	Null?	Type
UD_ADUSER_UID		VARCHAR2(20)
UD_ADUSER_ORGNAME		VARCHAR2(256)
UD_ADUSRC_GROUPNAME		VARCHAR2(32)
OIM_BLKLD_SEQ	NOT NULL	NUMBER(19)

15.7.2 Determining Values for the Input Parameters of the Utility

The following are input parameters of the utility:

- Oracle Home

  Value of the ORACLE_HOME environment variable on the host computer for the Oracle Identity Manager database.

- Database Connection String

  Connection string to connect to the database that must be entered in the following format:

  //HOST_IP_ADDRESS:PORT_NUMBER/SERVICE_NAME

- OIM DB User

  Database login ID of the Oracle Identity Manager database user.

- OIM DB Pwd

  Password of the Oracle Identity Manager database user. This must be entered twice when prompted.

- Application instance name (APP_INSTANCE)

  Name of the application instance corresponding to the account data to be loaded. If the user is not aware of the application instance name, then Account Bulkload utility prompts for the resource object name and IT resource name. The prompt is as shown:

  ```
  Do you know the Application Instance name? (Y,y,N,n)
  ```

  If you enter Y or y, then you are prompted for the application instance name. If you enter N or n, then you are prompted for the following:

  - Resource Object Name (OBJ_NAME)

    If the user is not aware of the application instance name, then Bulk Load utility prompts for the resource object name corresponding to the account data to be loaded.

  - IT Resource Name

    Name of the IT resource created for the target system. This is required only when the user is not aware of the application instance name. The account bulkload utility first prompts for resource object name, and then prompts for IT resource name.

- CSV file names

  Names of the CSV files to be used as the input source.

  This parameter is used only if the input source is CSV files. See "Using CSV Files As the Input Source" on page 15-21 for more information. If you are loading data from parent and child CSV file, then use a comma-delimited list to enter the names of the files. The name of the parent CSV file must be provided first, and it must be followed by the names of the child CSV files. In addition, enter the column that links the parent and child data.

- Tmp table name

  Name of the temporary table to be used as the input source.

  This parameter is used only if the input source for the bulk load operation is a database table. See "Creating Database Tables As the Input Source" on page 15-22 for more information.

- Control Line

  Comma-separated list of names of columns to be loaded from the database table into Oracle Identity Manager.

  This parameter is used only if the input source for the bulk load operation is a database table.

- Tablespace Name

  Name of the tablespace in which temporary tables are to be created during the bulk load operation (if end user won't provide the tablespace name then it will pick the default tablespace).

  See "Preparing Your Database for a Bulk Load Operation" on page 15-6 for more information.

- Date format

  Date format used by date columns in the CSV files.

  This parameter is used only if the input source is a single or multiple CSV files.

  The date format must match the following:

  - Oracle supported date formats, such as dd-mm-yyyy or MM-DD-YYYY
  - The date format specified in the CSV file

- Batch Size

  Number of user records that must be processed by the utility as a single transaction.

The batch size can influence the performance of the bulk load operation. The default value of this parameter is 10000.

- Debug Flag

  You can specify Y or N as the value of this parameter. If this parameter is set to Y, then the utility records detailed information about events that occur during the bulk load operation. See "Data Recorded During the Operation" on page 15-42 for more information.

- Application Instance (APP_INSTANCE)

  Name of the application instance corresponding to the account data to be loaded.

  If the user is not aware of the application instance name, then account bulkload utility prompts for the Object name (OBJ_NAME)

- User ID (USR_LOGIN)

  The user login ID that is used to determine the user that provisioned Accounts using Bulk Load utility.

15.7.3 Monitoring the Progress of the Operation

During the bulk load operation, you can query the OIM_BLKLD_LOG table for information about the progress of the operation. For example, you can run the following query to see progress messages generated during the bulk load operation to load account data:

```
SELECT MSG FROM OIM_BLKLD_LOG
WHERE MODULE = 'ACCOUNT' AND LOG_LEVEL = 'PROGRESS_MSG'
ORDER BY MSG_SEQ_NO;
```

Errors encountered during the bulk load operation can be viewed by querying the OIM_BLKLD_LOG table. The following is an example of the query to retrieve error messages:

```
SELECT MSG FROM OIM_BLKLD_LOG
WHERE MODULE = 'ACCOUNT' AND LOG_LEVEL = 'ERROR'
ORDER BY MSG_SEQ_NO;
```

15.7.4 Handling Exceptions Recorded During the Operation

At the end of a bulk load operation, the utility records statistics related to the operation in the following file:

oimbulkload/logs_*YYYYMMDD_hhmm*/oim_blkld_account_load_summary.log

To determine if there were exceptions during the operation, open this log file and look for the number against the Number of Records Rejected label. If the number of rejected records is greater than zero, then exceptions were thrown during the operation. User records that are rejected by the utility are recorded in the exception table (OIM_BLKLD_EX_*SUFFIX*). For each rejected record, the EXCEPTION_MSG column in the OIM_BLKLD_EX_*SUFFIX* table stores information about the reason the record could not be loaded.

Example 15–2 shows sample statistics recorded in the log file at the end of a bulk load operation to store account data.

Example 15–2 Sample Log File Generated After Loading Account Data

```
=============================================================
A C C O U N T   L O A D   S T A T I S T I C S
=============================================================
Start Time:   22-JUL-08 03.59.30.206000 PM
End Time:     22-JUL-08 04.03.21.126000 PM
Number of Records Processed:  100026
Number of Records Loaded:     100000
Number of Records Rejected:   26
=============================================================

The names of the TMP tables used during the load:
OIM_BLKLD_TMP_P100001
OIM_BLKLD_TMP_C100002
The names of the Exception tables used during the load:
OIM_BLKLD_EX_P100001
OIM_BLKLD_EX_C100002
```

In this sample, the number of rejected records is 26. If the log file shows that any records were rejected by the utility, then see "Fixing Exceptions and Reloading Data Records" for information about retrying the load operation for these records.

Note: At the end of each bulk load operation, it is recommended that you create a backup of the exception tables.

15.7.5 Fixing Exceptions and Reloading Data Records

Note: If you want to load data from CSV files for multiple target systems, then you can apply one of the following approaches:

- **Approach 1:** Run the utility for all the sets of CSV files, and then perform the procedure described in this section.
- **Approach 2:** Run the utility for one set of CSV files, and perform the procedure described in this section. Then, repeat this procedure for the next set of CSV files.

As mentioned earlier, errors encountered during the bulk load operation can be viewed by querying the OIM_BLKLD_LOG table. The following is an example of the query to retrieve error messages:

```
SELECT MSG FROM OIM_BLKLD_LOG
WHERE MODULE = 'ACCOUNT' AND LOG_LEVEL = 'ERROR'
ORDER BY MSG_SEQ_NO;
```

An exception table OIM_BLKLD_EX_*SUFFIX* is created for each data table used as the input source during the bulk load operation. Records that do not meet the criteria for the operation are copied into this exception table. The suffix appended to the name of each exception table is the same as suffix appended to the name of the corresponding data table.

To reload rejected records:

1. Create a backup of the exception table in which rejected records are stored.

Note: Although this is an optional step, it is recommended that you create a backup.

2. Review each record in the exception table, and fix errors in the data based on the message recorded in the EXCEPTION_MSG column.
3. After you fix errors in all the rejected records in an exception table, rename the table to OIM_BLKLD_TMP_*SUFFIX* and then use it as the input source.
4. Load records from the OIM_BLKLD_TMP_*SUFFIX* table by running the utility. See "Running the Utility" for more information.
5. Repeat Steps 1 through 4 until the Number of Records Rejected label in the oim_blkld_account_load_summary.log file shows the value 0.
6. Restart Oracle Identity Manager if loading was done in offline mode.

Note: Being a database-intensive operation by design, Bulk Load disables the constraints and indexes on the relevant Oracle Identity Manager entity tables during the start of the operation. Bulk Load operation failure towards the end of the load might at times render the indexes and constraints in disabled state. To identify and fix this issue, manually restore the indexes and constraints as follows:

1. Identify the unusable indexes and disabled constraints. To do so, the following SQL queries or similar mechanism can be used:

   ```
   SELECT TABLE_NAME, CONSTRAINT_NAME FROM user_constraints WHERE
   status = 'DISABLED';
   SELECT index_name FROM user_indexes WHERE status = 'UNUSABLE';
   ```

2. Enable the constraints and rebuild the indexes manually, as shown:

   ```
   ALTER TABLE TABLE_NAME ENABLE CONSTRAINT CONSTRAINT_NAME;
   ALTER INDEX INDEX_NAME REBUILD;
   ```

15.7.6 Verifying the Outcome of the Bulk Load Operation

To verify the outcome of the bulk load operation, check if you are able to perform the following steps for one of the OIM Users for whom an account has been added by the utility:

- Log in as the OIM User, and check if the newly created account is displayed in the Accounts tab of the User details page or in My Accounts tab of My Access page for the user.
- Log in to the target system by using the credentials of the newly created account.

15.8 Loading Role, Role Hierarchy, Role Membership, and Role Category Data

The following is a summary of the steps involved in loading role-related data:

1. Prepare your database for a bulk load operation, if not already done. See "Preparing Your Database for a Bulk Load Operation" on page 15-6 for details.
2. Create the input source for the bulk load operation.

   If you want to use a database table as the input source, then create the table and copy role-related data into the table.

If you want to use CSV files as the input source, then create the CSV files and copy role-related data into the files. In addition, create a master.txt file containing the names of the files in the sequence in which you want to load data from them.

3. Determine values for the input parameters of the utility.
4. Stop Oracle Identity Manager if you want to run in offline mode. For online mode, Oracle Identity Manager server can be running.
5. Run the oim_blkld.sh or oim_blkld.bat script.
6. Monitor the progress of the bulk load operation.
7. Determine the outcome of the bulk load operation.
8. If required, reload data that is not loaded during the first run.
9. Restart Oracle Identity Manager if it was stopped in step 4.
10. Verify the outcome of the bulk load operation.
11. Gather diagnostic data from the operation.
12. Remove temporary tables and files created during the operation.

The following sections provide detailed information about the steps involved in loading OIM User data:

- Creating the Input Source for the Bulk Load Operation
- Determining Values for the Input Parameters of the Utility
- Monitoring the Progress of the Operation
- Handling Exceptions Recorded During the Operation
- Fixing Exceptions and Reloading Data Records
- Verifying the Outcome of the Bulk Load Operation

15.8.1 Creating the Input Source for the Bulk Load Operation

Depending on the input source that you want to use, apply the guidelines given in one of the following sections:

- Using CSV Files As the Input Source
- Creating Database Tables As the Input Source
- Determining the UGP_NAME Generated After Role Load

15.8.1.1 Using CSV Files As the Input Source

If you want to use CSV files as the input source for the bulk load operation, then apply the following guidelines while creating the CSV files:

- The CSV files must be placed in the oimbulkload/csv_files directory.
- The first line in the CSV file is called the control line.
- This line must contain a comma-separated list of column names based on the selected role upload (role, role hierarchy, role membership, and role category) in the Oracle Identity Manager database.
- From the second line onward, the file must contain values for the columns in the control line. The order of columns in the first line and the values in the rest of the lines must be the same. The following is a sample content of a role (UGP) CSV file:

```
UGP_ROLENAME,UGP_NAMESPACE,USR_LOGIN,ORG_NAME,INCLUDE_HIERARCHY
"Finance Controllers",Default,XELSYSADM,Finance,YES
"Finance Controllers",Default,XELSYSADM,Requests,YES
```

- Role load is capable of publishing the roles to organizations to follow the security model in Oracle Identity Manager, with an option to include hierarchy.

  As a value of the ORG_NAME parameter, specify the organization name, such as Finance or Requests, to which you want to publish the roles. Specify YES for INCLUDE_HIERARCHY if you want to publish the roles to the specified organization and its suborganizations. Specify NULL or NO for INCLUDE_HIERARCHY if you want to publish the roles only to the specified organization and not its suborganizations. If you do not specify values for the ORG_NAME and INCLUDE_HIERARCHY parameters, then by default, the roles are published to the Top organization with hierarchy.

- If the value in any column contains a comma, then that value must be enclosed in double quotation marks (").
- The CSV file must contain values for all columns that are designated as mandatory in the respective role tables.
- The CSV file must contain values for all columns that are designated as mandatory depending on the upload role data, role hierarchy data, role membership data, and role category data.
  - Role UGP): UGP_ROLENAME,UGP_NAMESPACE,USR_LOGIN,ORG_NAME,INCLUDE_HIERARCHY (UGP_NAMESPACE,ORG_NAME)

    INCLUDE_HIERARCHY can be left as null when not required.
  - Role Hierarchy (GPG): UGP_NAME, GPG_UGP_NAME
  - Role Membership (USG): UGP_NAME, USR_LOGIN
  - Role Category (ROLE_CATEGORY): ROLE_CATEGORY_NAME

  Each row in the CSV file must have a unique value for the combination of mandatory columns.

- The following default values are inserted into Oracle Identity Manager if the CSV file does not contain values for these columns:
  - For Role (UGP)

    ROLE_CATEGORY_NAME: Default

    UGP_DISPLAY_NAME: Defaults to UGP_NAME

    ORG_NAME: TOP

    INCLUDE_HIERARCHY: YES
  - For Role Hierarchy (GPG)

    None
  - For Role Membership (USG)

    RUL_KEY: RUL_KEY from RUL table with RUL_NAME as 'Default'

    USG_PRIORITY: group and rank based on UGP_KEY based on the rows given for upload.
  - Role Category (ROLE CATEGORY)

None

- Create a master TXT file containing the names of the CSV files containing role data to be loaded. You can specify any name for the file, for example, master.txt. Save the master file in the oimbulkload/csv_files directory.

  If you want to load multiple CSV files, then enter the name of each data CSV file on a separate line in the master file. Order the list of CSV file names in the sequence in which you want the utility to load data from the files. For example, suppose you have created three data CSV files, Role1.csv, Role2.csv, and Role3.csv. In the master file, enter the names of the data CSV files in the following order:

  Role1.csv

  Role2.csv

  Role3.csv

  When you run the utility, data is loaded in this order.

- If the CSV file is generated on Microsoft Windows and is to be loaded on Linux environment, then remove the special characters, such as '\n\r', to avoid run-time errors.

15.8.1.2 Creating Database Tables As the Input Source

If you want to use a database table as the input source for loading OIM User data, then apply the following guidelines while creating the database table:

- Create the table in the Oracle Identity Manager database.
- The table must contain the following primary key column:

  ```
  OIM_BLKLD_USRSEQ NUMBER(19)
  ```

  The utility uses this column as the primary key. If required, you can use a database sequence to populate this column.

- The rest of the columns must be the same as the ones in the respective role tables that you want to use.

Table 15–4 shows the structure of a sample database role table.

Table 15–4 Structure of a Sample Database Table

Role	NULL	Type
UGP_ROLENAME	NOT NULL	VARCHAR2(2000)
UGP_NAMESPACE		VARCHAR2(512)
USR_LOGIN	NOT NULL	VARCHAR(256)
ORG_NAME	NOT NULL	VARCHAR2(256)
INCLUDE_HIERARCHY	NOT NULL	VARCHAR2(256)
...	...	...
OIM_BLKLD_USRSEQ	NOT NULL	NUMBER(19)

Note: ORG_NAME and INCLUDE_HIERARCHY are required for loading roles only, and not for role hierarchy, role membership, and role category.

15.8.1.3 Determining the UGP_NAME Generated After Role Load

Bulkload utility generates UGP_NAME during role load in the following format:

```
UGP_NAMESPACE.UGP_ROLENAME
```

By default, the value of UGP_NAMESPACE is Default, when you do not provide any specific value for UGP_NAMESPACE in the CSV file. To determine the generated UGP_NAME:

1. If UGP_NAMESPACE is null in the CSV file, then the namespace value is Default, and the generated UGP_NAME is equal to the value of UGP_ROLENAME.
2. If UGP_NAMESPACE is not null and has a defined value in the CSV file, then the generated UGP_NAME is equal to the value of UGP_NAMESPACE.UGP_ROLENAME.

On the basis of the UGP_NAME generation methodology, you can determine the UGP_NAME values for the next loading of role hierarchy, role membership, and role category, even if you do not have direct access to the database. Otherwise, you can check the generated value of UGP_NAME in the UGP table.

15.8.2 Determining Values for the Input Parameters of the Utility

The following are input parameters of the utility:

- Oracle Home

  Value of the ORACLE_HOME environment variable on the host computer for the Oracle Identity Manager database

- Database Connection String

  Connection string to connect to the database that must be entered in the following format:

  //HOST_IP_ADDRESS:PORT_NUMBER/SERVICE_NAME

- OIM DB User

  Database login ID of the Oracle Identity Manager database user

- OIM DB Pwd

  Password of the Oracle Identity Manager database user. Enter the password twice when prompted.

- CSV file names

  Names of the CSV files to be used as the input source

  This parameter is used only if the input source is CSV files. See "Using CSV Files As the Input Source" on page 15-28 for more information. If you are loading data from parent and child CSV file, then use a comma-delimited list to enter the names of the files. The name of the parent CSV file must be provided first, and it must be followed by the names of the child CSV files.

- Tmp table name

  Name of the temporary table to be used as the input source

  This parameter is used only if the input source for the bulk load operation is a database table. See "Creating Database Tables As the Input Source" on page 15-30 for more information.

- Control Line

Comma-separated list of names of columns to be loaded from the database table into Oracle Identity Manager

This parameter is used only if the input source for the bulk load operation is a database table.

- Tablespace Name

  Name of the tablespace in which temporary tables are to be created during the bulk load operation (if end user won't provide the tablespace name then it will pick the default tablespace)

  See "Preparing Your Database for a Bulk Load Operation" on page 15-6 for more information.

- Date format

  Date format used by date columns in the CSV files. This is prompted only for role load, and not for role hierarchy, role membership, and role category.

  This parameter is used only if the input source is a single or multiple CSV files.

  The date format must match the following:

  – Oracle supported date formats, such as dd-mm-yyyy or MM-DD-YYYY

  – The date format specified in the CSV file

- Batch Size

  Number of user records that must be processed by the utility as a single transaction

  The batch size can influence the performance of the bulk load operation. The default value of this parameter is 10000.

- Debug Flag

  You can specify Y or N as the value of this parameter. If this parameter is set to Y, then the utility records detailed information about events that occur during the bulk load operation. See "Data Recorded During the Operation" on page 15-42 for more information.

15.8.3 Monitoring the Progress of the Operation

During the bulk load operation, you can query the OIM_BLKLD_LOG table for information about the progress of the operation. For example, you can run the following query to see progress messages generated during the bulk load operation to load OIM Role data:

```
SELECT MSG FROM OIM_BLKLD_LOG
WHERE MODULE = 'ROLE' AND LOG_LEVEL = 'PROGRESS_MSG'
ORDER BY MSG_SEQ_NO;
```

Errors encountered during the bulk load operation can be viewed by querying the OIM_BLKLD_LOG table. The following is an example of the query to retrieve error messages:

```
SELECT MSG FROM OIM_BLKLD_LOG
WHERE MODULE = 'ROLE' AND LOG_LEVEL = 'ERROR'
ORDER BY MSG_SEQ_NO;
```

15.8.4 Handling Exceptions Recorded During the Operation

At the end of a bulk load operation, the utility records statistics related to the operation in the following file:

oimbulkload/logs_*YYYYMMDD_HHMM*/oim_blkld_*ENTITY_NAME*_load_summary.log

In the log file name, *ENTITY_NAME* stands for the entity being loaded. For example:

- For roles, the log file name is oim_blkld_role_load_summary.log.
- For role memberships, the log file name is oim_blkld_rolemem_load_summary.log.

To determine if there were exceptions during the operation, open this log file and look for the number against the Number of Records Rejected label. If the number of rejected records is greater than zero, then exceptions were thrown during the operation. User records that are rejected by the utility are recorded in the exception table (OIM_BLKLD_EX_SUFFIX). For each rejected record, the EXCEPTION_MSG column in the OIM_BLKLD_EX_SUFFIX table stores information about the reason the record could not be loaded.

Example 15–3 shows sample statistics recorded in the log file at the end of a bulk load operation to store OIM Role data.

Example 15–3 Sample Log File Generated After Loading OIM Role Data

```
*********************************************************************************
*****************
Processing File: Role.csv
=================================================================================
========
R O L E   L O A D   S T A T I S T I C S   F O R   F I L E : Role.csv
=================================================================================
========
Start Time:   17-NOV-09 02.48.18.447767 AM
End Time:     17-NOV-09 02.48.19.228710 AM
Number of Records Processed:  2
Number of Records Loaded:     2
Number of Records Rejected:   0
=================================================================================
========

The name of the TMP table used during the load:
OIM_BLKLD_TMP_ROLE1

The name of the Exception table used during the load:
OIM_BLKLD_EX_ROLE1
=================================================================================
========
==============================================================================
Time taken in re-building indexes and enabling FK constraints
==============================================================================

Start time:      17-NOV-09 02.48.19.243781 AM
```

In this sample, the number of rejected loaded is 2. If the log file shows that any records have been rejected by the utility, then see "Fixing Exceptions and Reloading Data Records" on page 15-34 for information about retrying the load operation for these records.

Note: You cannot use the utility to load data into a remote Oracle Identity Manager database.

15.8.5 Fixing Exceptions and Reloading Data Records

As mentioned earlier, errors encountered during the bulk load operation can be viewed by querying the OIM_BLKLD_LOG table. The following is an example of the query to retrieve error messages:

```
SELECT MSG FROM OIM_BLKLD_LOG
WHERE MODULE = 'ROLE' AND LOG_LEVEL = 'ERROR'
ORDER BY MSG_SEQ_NO;
```

An exception table OIM_BLKLD_EX_SUFFIX is created for each data table used as the input source during the bulk load operation. Records that do not meet the criteria for the operation are copied into this exception table. The suffix appended to the name of each exception table is the same as suffix appended to the name of the corresponding data table.

To reload rejected records:

1. Create a backup of the exception table in which rejected records are stored.

   Note: Although this is an optional step, it is recommended that you create a backup.

2. Review each record in the exception table, and fix errors in the data based on the message recorded in the EXCEPTION_MSG column.
3. After you fix errors in all the rejected records in an exception table, rename the table to OIM_BLKLD_TMP_SUFFIX and then use it as the input source.
4. Load records from the OIM_BLKLD_TMP_SUFFIX table by running the utility. See "Running the Utility" on page 15-6 for more information.
5. Repeat Steps 1 through 4 until the Number of Records Rejected label shows the value 0 in the oim_blkld_role_load_summary.log file or the corresponding log file for role membership, role hierarchy, and role category.
6. Restart Oracle Identity Manager if loading was done in offline mode.

Note: Being a database-intensive operation by design, Bulk Load disables the constraints and indexes on the relevant Oracle Identity Manager entity tables during the start of the operation. Bulk Load operation failure towards the end of the load might at times render the indexes and constraints in disabled state. To identify and fix this issue, manually restore the indexes and constraints as follows:

1. Identify the unusable indexes and disabled constraints. To do so, the following SQL queries or similar mechanism can be used:

   ```
   SELECT TABLE_NAME, CONSTRAINT_NAME FROM user_constraints WHERE
   status = 'DISABLED';
   SELECT index_name FROM user_indexes WHERE status = 'UNUSABLE';
   ```

2. Enable the constraints and rebuild the indexes manually, as shown:

   ```
   ALTER TABLE TABLE_NAME ENABLE CONSTRAINT CONSTRAINT_NAME;
   ALTER INDEX INDEX_NAME REBUILD;
   ```

15.8.6 Verifying the Outcome of the Bulk Load Operation

To verify the outcome of the bulk load operation, check if you are able to perform the following steps for one of the OIM Role added by the utility:

1. Log in to Oracle Identity Self Service, and verify that the newly created role is displayed in the search result for roles.
2. For the newly created role hierarchy and role members, click the **Hierarchy** and **Members** tabs respectively on the role details page.
3. To verify the newly created role category, in the Welcome page of Oracle Identity Administration, click **Advanced Search - Role Categories**. Then, perform an advanced search to find the newly created role.

15.9 Loading Organization Data

The following is a summary of the steps involved in loading organization data:

1. Prepare your database for a bulk load operation, if not already done. See "Preparing Your Database for a Bulk Load Operation" on page 15-6 for details.
2. Create the input source for the bulk load operation.

   If you want to use a database table as the input source, then create the table and copy organization-related data into the table.

   If you want to use CSV files as the input source, then create the CSV files and copy organization-related data into the files. In addition, create a master.txt file containing the names of the files in the sequence in which you want to load data from them.
3. Determine values for the input parameters of the utility.
4. Stop Oracle Identity Manager if you want to run Bulk Load utility in offline mode. For online mode, Oracle Identity Manager server can be running.
5. Run the `oim_blkld.sh` (for UNIX) or `oim_blkld.bat` (for Windows) script.
6. Monitor the progress of the bulk load operation.
7. Determine the outcome of the bulk load operation.

8. If required, reload data that is not loaded during the first run.
9. Restart Oracle Identity Manager , if it was stopped in step 4.
10. Verify the outcome of the bulk load operation.
11. Gather diagnostic data from the operation.
12. Remove temporary tables and files created during the operation.

The following sections provide detailed information about the steps involved in loading OIM Organization data:

- Creating the Input Source for the Bulk Load Operation
- Determining Values for the Input Parameters of the Utility
- Monitoring the Progress of the Operation
- Handling Exceptions Recorded During the Operation
- Fixing Exceptions and Reloading Data Records
- Verifying the Outcome of the Bulk Load Operation

15.9.1 Creating the Input Source for the Bulk Load Operation

Depending on the input source that you want to use, apply the guidelines given in one of the following sections:

- Using CSV Files as the Input Source
- Creating Database Tables as the Input Source

15.9.1.1 Using CSV Files as the Input Source

If you want to use CSV files as the input source for the bulk load operation, then apply the following guidelines while creating the CSV files:

- The CSV files must be placed in the `oimbulkload/csv_files` directory.
- The first line in the CSV file is called the control line. This line must contain a comma-separated list of column names of the ACT table in the Oracle Identity Manager database.
- From the second line onward, the file must contain values for the columns in the control line. The order of columns in the first line and the values in the rest of the lines must be the same. The following are sample contents of a CSV file:

```
ACT_NAME,ACT_PARENT_NAME,ACT_STATUS,ACT_CUST_TYPE
Org1,Xellerate Users,Active,System
Org2, Org3,Active,Company
Org3,Org4,Active, System
Org4,Top,Active,Company
```

- The CSV file can contain hierarchal data as well, such as the example provided in the previous bullet point. Here, Org4 is created first, then Org3, and finally Org2 is created. If the last entry of Org4 is missed from this CSV, then neither Org3 nor Org2 are created because respective parents are not available in Oracle Identity Manager.
- ACT_NAME and ACT_PARENT_NAME are mandatory columns .Along with these columns you can also load other columns present in ACT table.
- If the value in any column contains a comma, then that value must be enclosed in double quotation marks (").

- The CSV file must contain values for all columns that are designated as mandatory in the ACT table.
- Each row in the CSV file must have a unique value for the ACT_NAME column in the USR table. If there are multiple files, then ensure that ACT_NAME values are unique across the CSV files. This check for uniqueness of ACT_NAME values must also cover existing organization in Oracle Identity Manager.
- Note that the following default values are inserted into Oracle Identity Manager if the CSV file does not contain values for these columns:

  ACT_PARENT_NAME : `Top`

  ACT_CUST_TYPE : `System`

  ACT_STATUS: `Active`
- Create a master TXT file containing the names of the CSV files containing organization data to be loaded. You can specify any name for the file, for example, master.txt. Save the master file in the `oimbulkload/csv_files` directory.

  If you want to load multiple CSV files, then enter the name of each data CSV file on a separate line in the master file. Order the list of CSV file names in the sequence in which you want the utility to load data from the files. For example, suppose you have created three data CSV files, London_Orgs.csv, NewYork_Orgs.csv, and Tokyo_Orgs.csv. In the master file, you enter the names of the data CSV files in the following order:

  ```
  Tokyo_Orgs.csv
  London_Orgs.csv
  NewYork_Orgs.csv
  ```

  When you run the utility, data is loaded in this order. This is because the organization data in London and New York may have a dependency on the Tokyo Orgs.
- If the CSV file is generated on Microsoft Windows and is to be loaded on Linux environment, then remove the special characters, such as '\n\r', to avoid run-time errors.

Note: While copying a CSV file from Windows to UNIX, Solaris, or Linux systems, some special characters, such as `^M`, are appended to the file. This is because, the file from Windows is in DOS (ASCII) format and must be converted to ISO format.

Solaris preinstalls the `dos2unix` utility into the system to do this job. But for UNIX/Linux systems, the CSV file must be converted from DOS format to UNIX format to ensure sanity of the input file before being used in the Bulk Load operation. To do this, the syntax is:

```
# dos2unix CSV_FILE_NAME
```

If the `dos2unix` utility does not exist in the UNIX/Linux systems, then the administrator can install the utility for the respective UNIX/Linux versions by using the relevant documentation.

15.9.1.2 Creating Database Tables as the Input Source

If you want to use a database table as the input source for loading organization data, then apply the following guidelines while creating the database table:

- Create the table in the Oracle Identity Manager database.
- ACT_NAME and ACT_PARENT_NAME are mandatory columns.
- Along with mandatory columns, the table must contain the following primary key column:

  ```
  OIM_BLKLD_ACTSEQ NUMBER(19)
  ```

  The utility uses this column as the primary key. If required, you can use a database sequence to populate this column.
- The rest of the columns must be the same as the ones in the ACT table that you want to use. In other words, ignore optional ACT columns that you do not want to include in the table that you create.

Table 15–5 lists the structure of a sample database Org table.

Table 15–5 Structure of a Sample Database Table

Column	Null	Type
OIM_BLKLD_ACTSEQ		NUMBER(19)
ACT_NAME	NOT NULL	VARCHAR2(256 CHAR)
ACT_PARENT_NAME		VARCHAR2(256 CHAR)
ACT_CUST_TYPE		VARCHAR2(256 CHAR)

Note: The following default values are inserted into Oracle Identity Manager if the table does not contain values for these columns:

ACT_PARENT_NAME : `Top`

ACT_CUST_TYPE : `System`

ACT_STATUS: `Active`

15.9.2 Determining Values for the Input Parameters of the Utility

The following are input parameters of the utility:

- Oracle Home

  Value of the ORACLE_HOME environment variable on the host computer for the Oracle Identity Manager database
- Database Connection String

  Connection string to connect to the database that must be entered in the following format:

  //HOST_IP_ADDRESS:PORT_NUMBER/SERVICE_NAME
- OIM DB User

  Database login ID of the Oracle Identity Manager database user
- OIM DB Pwd

  Password of the Oracle Identity Manager database user. Enter the password twice when prompted.
- CSV file names

Names of the master CSV files to be used as the input source

This parameter is used only if the input source is CSV files. See "Using CSV Files As the Input Source" on page 15-28 for more information. If you are loading data from parent and child CSV file, then use a comma-delimited list to enter the names of the files. The name of the parent CSV file must be provided first, and it must be followed by the names of the child CSV files.

- Tmp table name

  Name of the temporary table to be used as the input source

  This parameter is used only if the input source for the bulk load operation is a database table. See "Creating Database Tables As the Input Source" on page 15-30 for more information.

- Control Line

  Comma-separated list of names of columns to be loaded from the database table into Oracle Identity Manager

  This parameter is used only if the input source for the bulk load operation is a database table.

- Tablespace Name

  Name of the tablespace in which temporary tables are to be created during the bulk load operation. (If end user does not provide the tablespace name, then it will pick the default tablespace.)

  See "Preparing Your Database for a Bulk Load Operation" on page 15-6 for more information.

- Date format

  Date format used by date columns in the CSV files.

  This parameter is used only if the input source is CSV file.

  The date format must match the following:

  – Oracle supported date formats, such as dd-mm-yyyy or MM-DD-YYYY

  – The date format specified in the CSV file

- Batch Size

  Number of user records that must be processed by the utility as a single transaction

  The batch size can influence the performance of the bulk load operation. The default value of this parameter is 10000.

- Debug Flag

  You can specify Y or N as the value of this parameter. If this parameter is set to Y, then the utility records detailed information about events that occur during the bulk load operation. See "Data Recorded During the Operation" on page 15-42 for more information.

15.9.3 Monitoring the Progress of the Operation

During the bulk load operation, you can query the OIM_BLKLD_LOG table for information about the progress of the operation. For example, you can run the following query to see progress messages generated during the bulk load operation to load OIM Organization data:

```
SELECT MSG FROM OIM_BLKLD_LOG
WHERE MODULE = 'ORG' AND LOG_LEVEL = 'PROGRESS_MSG'
ORDER BY MSG_SEQ_NO;
```

Errors encountered during the bulk load operation can be viewed by querying the OIM_BLKLD_LOG table. The following is an example of the query to retrieve error messages:

```
SELECT MSG FROM OIM_BLKLD_LOG
WHERE MODULE = 'ORG' AND LOG_LEVEL = 'ERROR'
ORDER BY MSG_SEQ_NO;
```

15.9.4 Handling Exceptions Recorded During the Operation

At the end of a bulk load operation, the utility records statistics related to the operation in the following file:

oimbulkload/logs_*YYYYMMDD_hhmm*/oim_blkld_org_load_summary.log

To determine if there were exceptions during the operation, open this log file and look for the number against the Number of Records Rejected label. If the number of rejected records is greater than zero, then exceptions were thrown during the operation. Organization records that are rejected by the utility are recorded in the exception table (OIM_BLKLD_EX_SUFFIX). For each rejected record, the ACT_LOAD_NOTE column in the OIM_BLKLD_EX_SUFFIX table stores information about the reason the record could not be loaded. ACT_LOAD_NOTE column in respective TMP table also shows the status/error of org creation.

Example 15-4 shows sample statistics recorded in the log file at the end of a bulk load operation to store OIM Organization data.

Example 15–4 Sample Log File Generated After Loading OIM Organization Data

```
//Sample log file when source of input is DB table-
*********************************************************************************
*****************
Bulkload Mode : online


*********************************************************************************
*****************
Source for Organization bulkload  : DB Table
Processing TMP table             : DB_TBL
successfully loaded  TMP Table    : DB_TBL
successfully loaded  TMP Table    : DB_TBL
TMP       Table                  : DB_TBL
Exception Table                  : DB_TBL_EX1
Log       Table                  : OIM_BLKLD_LOG

=================================================================================
========
O R G   L O A D   S T A T I S T I C S   F O R   T A B L E : DB_TBL
=================================================================================
========
Start Time                     :  19-FEB-16 12.55.35.101932 AM
End Time                       :  19-FEB-16 01.01.38.269610 AM
Number of Records Processed    :  160
Number of Records Loaded       :  160
Number of Records Rejected     :  0
```

```
//Sample log file when source of input is csv file-

*********************************************************************************
*****************
Bulkload Mode : online


*********************************************************************************
*****************
Source for Organization bulkload  : CSV File
Processing csv File               : org.csv
successfully loaded  File         : org.csv
successfully loaded  TMP Table    : OIM_BLKLD_TMP_ORG1
TMP       Table                   : OIM_BLKLD_TMP_ORG1
Exception Table                   : OIM_BLKLD_EX_ORG1
Log       Table                   : OIM_BLKLD_LOG

=================================================================================
========
O R G   L O A D   S T A T I S T I C S   F O R   F I L E : org.csv
=================================================================================
========
Start Time                     :  19-FEB-16 12.39.27.469318 AM
End Time                       :  19-FEB-16 12.39.27.519390 AM
Number of Records Processed    :  4
Number of Records Loaded       :  0
Number of Records Rejected     :  4
=================================================================================
========
```

In this sample, the number of rejected records is 4. If the log file shows that any records have been rejected by the utility, then see "Fixing Exceptions and Reloading Data Records" on page 15-41 for information about retrying the load operation for these records.

15.9.5 Fixing Exceptions and Reloading Data Records

As mentioned earlier, errors encountered during the bulk load operation can be viewed by querying the OIM_BLKLD_LOG table. The following is an example of the query to retrieve error messages:

```
SELECT MSG FROM OIM_BLKLD_LOG
WHERE MODULE = 'ORG' AND LOG_LEVEL = 'ERROR'
ORDER BY MSG_SEQ_NO;
```

An exception table OIM_BLKLD_EX_SUFFIX is created for each data table used as the input source during the bulk load operation. Records that do not meet the criteria for the operation are copied into this exception table. The suffix appended to the name of each exception table is the same as suffix appended to the name of the corresponding data table.

To reload rejected records:

1. Create a backup of the exception table in which rejected records are stored.

   Note: Although this is an optional step, it is recommended that you create a backup.

2. Review each record in the exception table, and fix errors in the data based on the message recorded in the EXCEPTION_MSG column.
3. After you fix errors in all the rejected records in an exception table, rename the table to OIM_BLKLD_TMP_SUFFIX, and run the following update statement:

   ```
   UPDATE OIM_BLKLD_TMP_SUFFIX
               SET   ACT_LOAD_STATUS='P',ACT_ACT_KEY=NULL,
                      ACT_LOAD_NOTE=NULL;
               COMMIT;
   ```

   Now use the table as the input source.
4. Load records from the OIM_BLKLD_TMP_SUFFIX table by running the utility. See "Running the Utility" on page 15-6 for more information.
5. Repeat Steps 1 through 4 until the Number of Records Rejected label shows the value 0 in the oim_blkld_role_load_summary.log file.
6. Restart Oracle Identity Manager if loading has been in offline mode.

Note: Being a database-intensive operation by design, Bulk Load disables the constraints and indexes on the relevant Oracle Identity Manager entity tables during the start of the operation. Bulk Load operation failure towards the end of the load might at times render the indexes and constraints in disabled state. To identify and fix this issue, manually restore the indexes and constraints as follows:

1. Identify the unusable indexes and disabled constraints. To do so, the following SQL queries or similar mechanism can be used:

   ```
   SELECT TABLE_NAME, CONSTRAINT_NAME FROM user_constraints WHERE
   status = 'DISABLED';
   SELECT index_name FROM user_indexes WHERE status = 'UNUSABLE';
   ```

2. Enable the constraints and rebuild the indexes manually, as shown:

   ```
   ALTER TABLE TABLE_NAME ENABLE CONSTRAINT CONSTRAINT_NAME;
   ALTER INDEX INDEX_NAME REBUILD;
   ```

15.9.6 Verifying the Outcome of the Bulk Load Operation

To verify the outcome of the bulk load operation, check if you are able to perform the following steps for one of the OIM Organizations added by the utility:

1. Log in to Oracle Identity Self Service, and verify that the newly created organization is displayed in the search result for Organizations.
2. Create a user under newly created organization. To do so:
   a. Click the **Members** tab of the newly created organization. This user should be displayed as a member.
   b. Click the **Organization** tab of the user. The newly created organization should be displayed in the results.

15.10 Data Recorded During the Operation

During the bulk load operation, the utility inserts progress and error messages in the OIM_BLKLD_LOG table. Data in this table is not deleted at the start of a new bulk load operation. One of the columns in this table holds the time stamp at which messages are recorded in the table.

Table 15–6 describes the structure of the OIM_BLKLD_LOG table.

Table 15–6 Structure of the OIM_BLKLD_LOG Table

Column	NULL	Type	Description
MSG_SEQ_NO	NULL	NUMBER(19)	This column stores the number that denotes the order in which messages are inserted in this table. The column is populated by using the OIM_BLKLD_LOG_SEQ sequence. You can use this column to query for messages in the order in which they are recorded in the table.
MODULE	NOT NULL	VARCHAR2(20)	This column stores one of the following values: ROLE: This value indicates that the message has been recorded while loading OIM Role data. ROLE HIERARCHY: This value indicates that the message has been recorded while loading role hierarchy data. ROLE MEMBERSHIP: This value indicates that the message has been recorded while loading OIM role membership data. ROLE CATEGORY: This value indicates that the message has been recorded while loading OIM role category data.
LOG_LEVEL	NOT NULL	VARCHAR2(20)	This column stores one of the following values: **ERROR:** Designates fine-grained informational events that are useful to debug. **DEBUG:** Designates error events that might allow the application to continue running. Error is used to log all unhandled exceptions. **PROGRESS_MSG:** Designates intermediate progress messages.
LOAD_SOURCE	NOT NULL	VARCHAR2(40)	This column indicates the source of data for the bulk load operation during which the row was inserted. The value can be one of the following: CSV File: *FILE_NAME* DB Table

Table 15–6 (Cont.) Structure of the OIM_BLKLD_LOG Table

Column	NULL	Type	Description
MSG	NOT NULL	VARCHAR2(4000)	This column stores a message corresponding to the value stored in the LOG_LEVEL column.
CREATE_DATE		DATE	This column holds the time stamp at which the record was created. The format for entries in this column is as follows: yyyy/mm/dd hh24:mi:ss For example: 2008/06/23 21:49:16:32

15.11 Gathering Diagnostic Data from the Bulk Load Operation

As mentioned earlier in this document, the following log files are created during the bulk load operation:

- For OIM Users:

  oimbulkload/logs_*YYYYMMDD_HHMM*/oim_blkld_user_load_summary.log

- For accounts:

  oimbulkload/logs_*YYYYMMDD_HHMM*/oim_blkld_account_load_summary.log

- For roles, role hierarchies, memberships, and role categories:

  oimbulkload/logs_*YYYYMMDD_HHMM*/oim_blkld_*ENTITY_NAME*_load_summary.log

  In the log file name, *ENTITY_NAME* stands for the entity being loaded. For example:

  – For roles, the log file name is oim_blkld_role_load_summary.log.

  – For role memberships, the log file name is oim_blkld_rolemem_load_summary.log.

Data recorded in this file can be used to collate performance-related information about the bulk load operation. The following information can be collected after the bulk load operation:

- Start time
- Input source
- Number of records in the system before the load
- Number of records successfully loaded
- Number of records rejected
- Total time taken

You can use this information during future runs of the utility.

See Also: Table 15–6, " Structure of the OIM_BLKLD_LOG Table" for information about the log levels that stores error events

15.12 Cleaning Up After a Bulk Load Operation

If you do not want to save the results of a bulk load operation, then:

- Remove the OIM_BLKLD_TMP_*SUFFIX*, OIM_BLKLD_EX_*SUFFIX*, and OIM_BLKLD_LOG tables.
- Remove any files that you created or used during the operation.
- If you created a tablespace for the operation, then remove the tablespace.
- See "Gathering Diagnostic Data from the Bulk Load Operation" before you remove log files created in the logs_*timestamp* directory.

Note: At this point, you can restart Oracle Identity Manager if you have not already done so.

15.13 Bulk Load High Volume Strategy and Case Studies

For information about general best practices and few case studies about high-volume data load, see the technote titled *OIM 11G BulkLoad Utility Strategies & Case Studies* (**Doc ID 1959363.1**) in the My Oracle Support web site at:

https://support.oracle.com

16 Developing Scheduled Tasks

Oracle Identity Manager contains a set of predefined tasks that can be scheduled as job runs. An example is a password warning task that sends email to users for password expiration.

Oracle Identity Manager also provides the capability of creating your own scheduled tasks. You can create scheduled tasks according to your requirements if none of the predefined scheduled tasks fit your needs.

For example, you can configure a reconciliation run using a scheduled task that checks for new information on target systems periodically and replicates the data in Oracle Identity Manager.

This chapter explains how to create and implement your custom scheduled tasks. It contains these topics:

16.1 Overview of Task Creation

This section outlines the essential steps in creating scheduled tasks, and presents an example to illustrate the process. Subsequent sections provide details on each step.

16.1.1 Steps in Task Creation

The basic steps for configuring new scheduled tasks are as follows:

1. Review Oracle Identity Manager's predefined scheduled tasks to determine whether a custom task is necessary.

   For details about the predefined tasks, see "Managing Scheduled Tasks" in the *Oracle Fusion Middleware System Administrator's Guide for Oracle Identity Manager*.

2. Determine key features of the scheduled task, such as the task name and the parameters that control the actions performed by the task.

   For details, see Section 16.2, "Defining the Metadata for the Scheduled Task".

3. Add the task metadata to the scheduled task XML file.

   For details, see Section 16.3, "Configuring the Scheduled Task XML File".

4. Develop the scheduled task Java class.

   For details, see Section 16.4, "Developing the Scheduled Task Class".

5. Declare the new scheduled task as a plug-in.

   For details, see Section 16.5, "Configuring the Plug-in XML File".

6. Package the task files so that Oracle Identity Manager can locate the files and make the task available for jobs.

   For details, see Section 16.6, "Creating the Directory Structure for the Scheduled Task".

16.1.2 Example of Scheduled Task

To illustrate the steps in developing a scheduled task, use an example scheduled task that retrieves employee records belonging to the given department from a given IT resource.

In addition, our scheduled task should allow the user to specify the number of records to be retrieved and whether to include disabled records in the retrieval.

16.2 Defining the Metadata for the Scheduled Task

Each scheduled task contains the following metadata information:

- Name of the scheduled task
- Name of the Java class that implements the scheduled task
- Description
- Retry Interval
- (Optional) Parameters that the scheduled task accepts. Each parameter contains the following additional information:
  - Parameter Name
  - Parameter Data Type
  - Required/ Optional Parameter
  - Help Text

16.3 Configuring the Scheduled Task XML File

Configuring the scheduled task XML file involves updating the XML file that contains the definitions of custom scheduled tasks. This section describes how to update the task XML file with the details of the new custom scheduled task.

You can modify the task.xml file located in the /db namespace of Oracle Identity Manager MDS schema, or you can create a custom scheduled task file. If you create a custom file, then the file name must be the same as the scheduled task name, with the .xml extension. You must import the custom scheduled task file to the /db namespace of Oracle Identity Manager MDS schema.

See Also: Chapter 17, "Developing Plug-ins" for examples of plug-ins.

Note: The scheduled task XML file can be imported into MDS using the Oracle Enterprise Manager. In a clustered environment, having the file in MDS avoids the need to copy the file on each node of the cluster.

For details about importing files into MDS, see "Migrating User Modifiable Metadata Files" on page 24-1.

The elements in the XML file reflect the task parameters that you described in Section 16.2, "Defining the Metadata for the Scheduled Task".

Example 16–1 shows a sample XML code for the scheduled task described in the preceding paragraph. Note that all the parameters are declared to be required parameters in this example.

Example 16–1 Sample XML for a Scheduled Task

```
<scheduledTasks xmlns="http://xmlns.oracle.com/oim/scheduler">
    <task>
        <name>Test_scheduled_task</name>
        <class>oracle.iam.scheduler.TestScheduler</class>
        <description>Retrieve Employee Records For Given Department</description>
        <retry>5</retry>
        <parameters>
            <string-param required="true" encrypted="false" helpText="Name of the
department">Department Name</string-param>
            <number-param required="true" helpText="Number of Records to Be
Retrieved">Number of Records</number-param>
            <boolean-param required="false" helpText="Retrieve disabled employee
records?">Get Disabled Employees</boolean-param>
        </parameters>
    </task>
</scheduledTasks>
```

See Also: "Scheduled Task Configuration File" on page 16-6 for details about the elements in the scheduled task configuration file.

This is basically exporting the task.xml from MDS and then adding the required tags to it and importing it back into MDS.

Note: For a task defined in a plugin, the metadata XML is not required to be seeded to MDS. This can be included in the META-INF folder in the plugin ZIP file. For details, see "Creating the Directory Structure for the Scheduled Task" on page 16-5.

You must export the task.xml file from MDS, add the required tags to the file, and then import it back to MDS. See "Migrating User Modifiable Metadata Files" on page 24-1 for information about exporting and importing MDS files.

16.4 Developing the Scheduled Task Class

The next step is to create a Java class to execute the task whose metadata was defined in the XML file. The Java class that implements a scheduled task is known as a **scheduled task class.**

To develop a Java class for the scheduled task:

1. Create a Java class file that extends the `oracle.iam.scheduler.vo.TaskSupport` class and overrides the `execute()` method with processing logic based on your requirements. The Java class must also override the other abstract methods:

```
public HashMap getAttributes();
public void setAttributes();
```

2. Create a JAR file for the Java class that you created. Name the JAR such that you can readily associate this JAR with your custom scheduled task.

   The JAR file can contain the dependent classes of the Java class. You can also create a separate JAR file for the dependent classes and place it in the lib/directory.

3. Copy the JAR file into the lib/ directory.
4. Repeat Steps 1 through 3 for every Java class that you want to create.

16.5 Configuring the Plug-in XML File

You must configure the plugin.xml file in order to declare the scheduled task as a plug-in. See Chapter 17, "Developing Plug-ins" for more information about plug-ins.

Note: Oracle recommends creating one plugin.xml file for one scheduled task. This is because when the plugin is unregistered, the corresponding package is deleted.

To configure the plugin.xml file:

1. Create the plugin.xml file by using any text editor.

Note: Create the plugin.xml file only if no such file exists. If there are existing plugins, then add a new plugin element for the new plugin.

2. Specify the plug-in point for the scheduled task by changing the value of the `pluginpoint` attribute of the plugins element to `oracle.iam.scheduler.vo.TaskSupport`.

The following XML code block from the plugin.xml file shows the value entered within the plugins element:

```
<plugins pluginpoint="oracle.iam.scheduler.vo.TaskSupport">
```

Note: For scheduled tasks, the <plugins> element remains the same for all scheduled tasks.

---

3. Add a <plugin> element for each scheduled task that you are adding.

   To specify the class that implements the plug-in (in this case, the scheduled task), change the value of the `pluginclass` attribute of the plugin element to the name of the Java class that implements the scheduled task. The following XML code block from the plugin.xml file shows sample values entered within the plugin element:

   ```
   <plugin pluginclass= "oracle.iam.scheduler.TestScheduler" version="1.0.1"
   name="scheduler element"/>
   ```

   After modification, the plugin.xml file looks similar to the following:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <oimplugins xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <plugins pluginpoint="oracle.iam.scheduler.vo.TaskSupport">
   <plugin pluginclass= "oracle.iam.scheduler.TestScheduler"
   version="1.0.1" name="scheduler element">
   </plugin>
   </plugins>
   </oimplugins>
   ```

4. Save and close the plugin.xml file.

16.6 Creating the Directory Structure for the Scheduled Task

The final step in configuring the scheduled task is to create a plugin.zip file with the directory structure specified in Example 16–2. In this example, a single plug-in is being added, but there can be multiple plugins in the plugin.zip file. Scheduler requires that files be zipped in a particular structure and named according to a particular naming convention. This ensures that Oracle Identity Manager identifies the custom scheduled tasks and makes it available in Oracle Identity System Administration while creating jobs.

Example 16–2 Directory Structure for the Scheduled Task

```
plugin/
      lib/
      PLUGIN.JAR
      plugin.xml
      META-INF (optional)
            METADATA.xml
```

Note that:

- The XML file for the plug-in must be named plugin.xml.
- The lib/ directory must contain only .JAR files. The lib/ directory consists of JAR files that contains the classes implementing the plug-in logic and the dependent library JAR files. In most instances, this directory consists of a single JAR file with

the implementation of all the plug-ins that are specified in plugin.xml. See "Developing Plug-ins" on page 17-6 for information about the directory structure.

- The directory for the scheduled task must contain the following files:
  - XML for the plug-in
  - JAR files
- There is one plugin.zip file for all the plug-ins that you create.
- The META-INF folder is an optional folder, in which metadata (task definition) file can be stored. If this file is placed in the META-INF folder, then it is not required to be seeded in MDS.

> **Note:** If the task definition XML file is stored in the META-INF directory rather than seeded to MDS, then there is a limitation of exporting the scheduled task by using the Deployment Manager.

- If META-INF folder does not exist or if the metadata file is not placed in the META-INF folder, then seed the file to MDS.

In the preceding example, *CLASS_NAME*.JAR is the JAR file that you create in Section 16.4, "Developing the Scheduled Task Class".

After you create the plugin.zip file, if deploying in a clustered environment, register the plug-in to the database by using appropriate APIs. See "Registering and Unregistering Plug-ins By Using APIs" on page 17-7 for details about registering plug-ins to Oracle Identity Manager by using APIs.

> **Note:** The XML for the plug-in must be named plugin.xml. Ensure that the lib directory contains only JAR files.

16.7 Scheduled Task Configuration File

This section describes the structure and details of the XML file containing scheduler task definitions. It contains the following topics:

- Structure of the Scheduler XML File
- The scheduledTasks Element
- The task Element
- The name Element
- The class Element
- The description Element
- The retry Element
- The parameters Element
- The string-param Element
- The number-param Element
- The boolean-param Element

16.7.1 Structure of the Scheduler XML File

The following is a list of elements in the configuration XML file:

```
<scheduledTasks xmlns="http://xmlns.oracle.com/oim/scheduler">
    <task>
        <name>
        <class>
        <description>
        <retry>
        <parameters>
            <string-param>
            .....
            </string-param>

            <number-param>
            .......
            </number-param>

            <boolean-param>
            .......
            </boolean-param>
        </parameters>
    </task>
</scheduledTasks>
```

16.7.2 The scheduledTasks Element

The scheduledTasks element is the root element in XML used to define scheduled tasks.

Table 16–1 summarizes the properties of the scheduledTasks element.

Table 16–1 Properties of the scheduledTasks Element

Property	Value
Parent Element	NA
Attributes	The XML namespace is specified as an attribute of the scheduledTasks element as follows: <scheduledTasks xmlns="http://xmlns.oracle.com/oim/scheduler"> **Note:** The xmlns parameter is mandatory.
Child Elements	task
Number of Occurrences	One for each scheduled task XML file to be created.
Element Value	NA
Mandatory or Optional?	Mandatory

16.7.3 The task Element

The task element is the child element of the scheduledTasks element.

You use the task element to define a scheduled task. The task element contains information about the scheduled task, for example, the name, class, description, and retry count of the scheduled task.

Table 16–2 summarizes the properties of the task element.

Table 16–2 Properties of the task Element

Property	Value
Parent Element	scheduledTasks
Attributes	None
Child Elements	name, class, description, retry, and parameters
Number of Occurrences	One for each task to be created. **NOTE:** If you want to define more than one task in a single scheduled task XML file, you must use one task element for every scheduled task being defined.
Element Value	NA
Mandatory or Optional?	Mandatory

16.7.4 The name Element

The name element is the child element of the task element. The name element is used to specify the name of the scheduled task being created.

Table 16–3 summarizes the properties of the name element.

Table 16–3 Properties of the name Element

Property	Value
Parent Element	task
Attributes	None
Child Elements	None
Number of Occurrences	One
Element Value	Name of the scheduled task being created. **Note:** The name of the scheduled task must be unique.
Mandatory or Optional?	Mandatory

16.7.5 The class Element

The class element is a mandatory element and is the child element of the task element. You use the class element to specify the name of the Java class that runs the scheduled task.

Table 16–4 summarizes the properties of the class element.

Table 16–4 Properties of the class Element

Property	Value
Parent Element	task
Attributes	None
Child Elements	None
Number of Occurrences	One
Element Value	Name of the Java class that runs the scheduled task. See "Developing the Scheduled Task Class" on page 16-4 for information on developing a class for the scheduled task.
Mandatory or Optional?	Mandatory

16.7.6 The description Element

The description element is a mandatory element and is the child element of the task element. You can use the description element to provide a description of the task being created.

Table 16–5 summarizes the properties of the description element.

Table 16–5 Properties of the description Element

Property	Value
Parent Element	task
Attributes	None
Child Elements	None
Number of Occurrences	One
Element Value	Description of the task being created
Mandatory or Optional?	Mandatory

16.7.7 The retry Element

Table 16–6 summarizes the properties of the retry element.

Table 16–6 Properties of the retry Element

Property	Value
Parent Element	task
Attributes	None
Child Elements	None
Number of Occurrences	One
Element Value	Number of seconds the scheduler must wait before it tries to schedule the task again
Mandatory or Optional?	Mandatory

16.7.8 The parameters Element

If you want to specify parameters at run time that the scheduled task requires for a successful job run, you must use the parameters element. For example, you might create a scheduled task that requires the user to specify the number of records to be retrieved at run time.

The parameters specified within this element are displayed under the Parameters section on the Create Job page.

Table 16–7 summarizes the properties of the parameters element.

Table 16–7 Properties of the parameters Element

Property	Value
Parent Element	task
Attributes	None
Child Elements	string-param, number-param, boolean-param
Number of Occurrences	One

Table 16–7 (Cont.) Properties of the parameters Element

Property	Value
Element Value	NA
Mandatory or Optional?	Optional

16.7.9 The string-param Element

You can use the string-param element to specify the name of the field that can take a value of the string data type. In other words, the string-param element specifies a label for the field that can hold a value of the string data type.

Table 16–8 summarizes the properties of the string-param element.

Table 16–8 Properties of the string-param Element

Property	Value
Parent Element	parameters
Attributes	required, helpText, encrypted
Child Elements	None
Number of Occurrences	One for every parameter of the string data type
Element Value	Name of the string parameter
Mandatory or Optional?	Optional

As listed in Table 16–8, the string-param element contains the following attributes:

- required

  This is a mandatory attribute and it can take a value of either `true` or `false`.

  If the value of the required attribute is `true`, it is mandatory to enter a value for the parameter at run time.

  If the value of the required attribute is `false`, it is not mandatory to enter a value for the parameter at run time.

- helpText

  Use this attribute to specify the text that must appear at run time to help users know what to enter in the field. The text that is specified is usually the description of the field that is being created by the parameter.

- encrypted

  By default, it has a value of `false` and this can take a value of either `true` or `false`.

  If the value of the encrypted attribute is `true`, then the entered value for the parameter at run time is stored in encrypted form.

  If the value of the required attribute is `false`, then the entered value for the parameter at run time is stored in plain text.

16.7.10 The number-param Element

You can use the number-param element to specify the name of the field that can take a value of the long data type.

Table 16–9 summarizes the properties of the number-param element.

Table 16–9 Properties of the number-param Element

Property	Value
Parent Element	parameters
Attributes	required, helpText
Child Elements	None
Number of Occurrences	One for every parameter of the long data type
Element Value	Name of field that can hold a long data type
Mandatory or Optional?	Optional

The behavior and description of the require and helpText attributes for the number-param and string-param elements is the same. See "The string-param Element" on page 16-10 for information about the require and helpText attributes.

16.7.11 The boolean-param Element

You can use the boolean-param element to specify the name of the field that can take a value of the boolean data type.

Table 16–10 summarizes the properties of the boolean-param element.

Table 16–10 Properties of the boolean-param Element

Property	Value
Parent Element	parameters
Attributes	required, helpText
Child Elements	None
Number of Occurrences	One for every parameter of the boolean data type
Element Value	Name of field that can hold a boolean data type
Mandatory or Optional?	Optional

The behavior and description of the require and helpText attributes for the boolean-param element and the string-param element is the same. See "The string-param Element" on page 16-10 for information about the require and helpText attributes.

16.8 Best Practices for Creating Custom Scheduled Tasks

Table 16–11 provides the guidelines for using variables/constants for creating custom scheduled tasks:

Table 16–11 Variables and Constants for Creating Custom Scheduled Tasks

Type	Example	Stor/Retrieve Value From
Target system connection details	Hostname, port number, SSL	IT Resource/application instance
Target system configurations	Attribute mappings, Unique Attribute, User Object Class	Lookup
Scheduled job-specific variables/constants	Application Instance Name, IT Resource Name, File Path, Search Filter, Batch Size, Retries	Scheduled job

Table 16–11 (Cont.) Variables and Constants for Creating Custom Scheduled Tasks

Type	**Example**	**Stor/Retrieve Value From**
Scheduled job advanced configuration variables/constants	Attribute Mappings, Target system Date Format, Constants, Attribute Transformation Classes	Lookup
Oracle Identity Manager-specific system wide highly static configuration properties/constants/variables	Default Date Format, Default policy for username generation	System properties
Email notifications	Subject, Body, To, From	Email templates

16.9 Using the isStop() Method

When a job is stopped from the Scheduler section in Oracle Identity System Administration, the job does not stop and keeps running. To stop the scheduled task, you can perform the following:

If you have developed a custom scheduled task, then you can call the isStop() or isStopped() method at various stages inside the execute method. If this method returns true, then return from the execute method. If you have loops inside the execute method, then make sure that the isStop() or isStopped() method is called for each loop iteration.

In the execute method, add a check for getting the job status. This can be obtained by calling the isStopped() method of the com.thortech.xl.scheduler.tasks.SchedulerBaseTask class. If the isStopped() method returns TRUE, then return from the execute method without performing any execution for the scheduled task. The following is the code snippet for this:

```
if(isStopped())
     return;
```

If you develop a custom scheduled task by extending the TaskSupport class in Oracle Identity Manager 11*g* Release 1 (11.1.1) or 11*g* Release 2 (11.1.2), then call the isStop() method in the execute method.

If the custom scheduled task code is extending legacy com.thortech.xl.scheduler.tasks.SchedulerBaseTask class of Oracle Identity Manager Release 9.x, then call the isStopped() method in the execute method.

16.10 Monitoring Scheduled Jobs Performance using DMS

Dynamic Monitoring Service (DMS) can be used to view performance metrics. The `OIM_ScheduledJob` DMS metrics is present for monitoring the performance of scheduled jobs. It provides details, such as number of scheduled jobs run and average time taken by scheduled job. Details of the successful jobs come under the `execute` column while failed job details come under the `Failed_execute` column.

Part V

Custom Operations

This part contains chapters that describe how to develop customized operations in the Oracle Identity Manager.

It contains the following chapters:

17 Developing Plug-ins

This chapter describes the concepts related to plug-in and how to develop and use a plug-in in the following sections:

- Plug-ins and Plug-in Points
- Using Plug-ins in Deployments
- Plug-in Points
- Configuring Plug-ins
- Developing Custom Plug-ins
- Registering Plug-ins
- Migrating Plug-ins

17.1 Plug-ins and Plug-in Points

A *plug-in* is a logical component that extends the functionality of features provided by Oracle Identity Manager. The plug-in framework enables you to define, register, and configure plug-ins, which extend the functionality provided by features. Plug-ins can be predefined or custom-developed, and they are utilized at plug-in points. A *plug-in point* is a specific point in the business logic where extensibility can be provided. An interface definition called the plug-in interface accompanies such a point. You can extend the plug-in interface based on the business requirements and register them as plug-ins. To do this, you develop a Plugin Java class and compile it before archiving in a JAR file, define plug-in metadata in an XML file, and ZIP these artifacts as a plug-in package that is ready to deploy.

For example, user creation is a business operation in Oracle Identity Manager. But this operation exposes a plug-in point for user name generation. If you want to model your custom logic of user name generation, then you must identify the plug-in point specifications and develop a plug-in accordingly.

The concepts related to plug-ins are described in the following sections:

- Plug-ins and Event Handlers
- Plug-in Stores

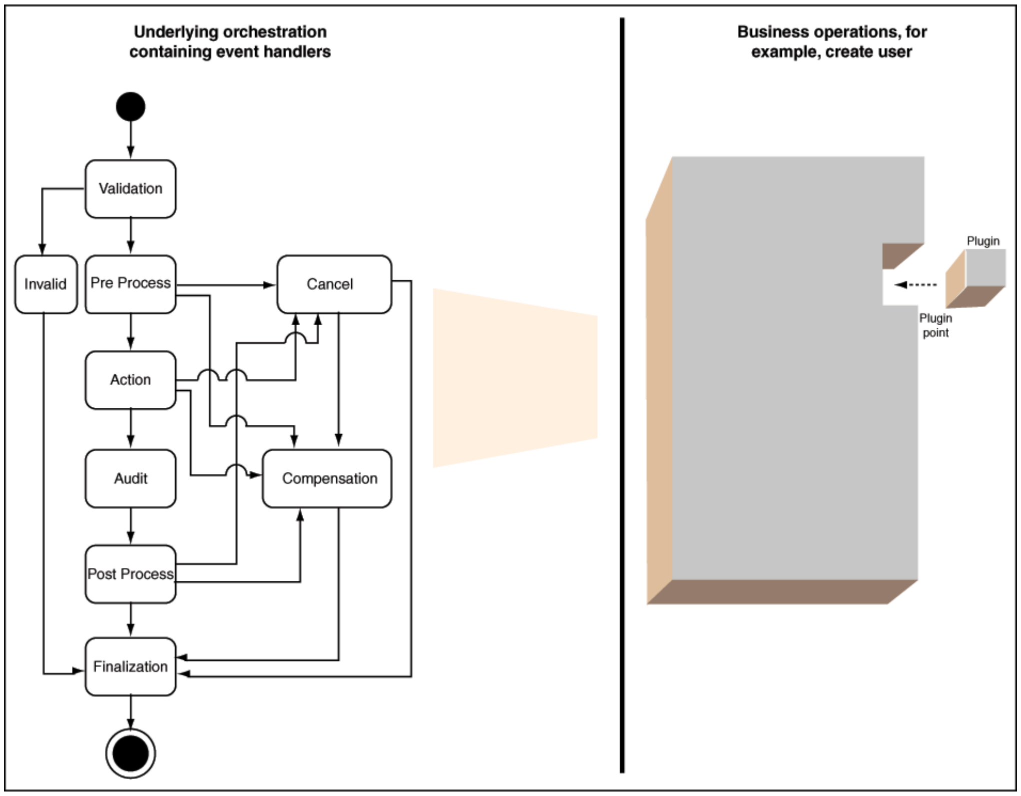
17.1.1 Plug-ins and Event Handlers

Most of the business operations in Oracle Identity Manager, such as user creation, role assignment to user, and user activation, are executed as orchestrations. Therefore, if there is a requirement to induce any custom logic in these operations or orchestrations, then you can model that logic as event handlers at stages, such as validation, preprocess, and postprocess, in which customization is supported.

However, you can analyze if any such operation also exposes a plug-in point for inducing the custom logic. If a plug-in point is available, then you can utilize the plug-in point rather than operating the underlying orchestration. For example, you can implement username generation by using the exposed plug-in without writing that as an event handler in the create user orchestration.

Figure 17–1 shows a diagrammatic representation of plug-ins and event handlers.

Figure 17–1 Plug-ins and Event Handlers



17.1.2 Plug-in Stores

The plug-in framework can store plug-ins in two types of stores:

- File system
- The Oracle Identity Manager database

When looking for plug-ins, the framework first examines plug-ins registered in the database, and looks in the file system.

17.1.2.1 File Store

The File Store consists of one or more directories on the Oracle Identity Manager host and is primarily used in development environments. This type of store is not appropriate for a production environment. File storage is convenient for the developer since there is no need to explicitly register the developed plug-ins with a file store.

Users can just drop in the plug-in zips or exploded plug-in directory to the designated location(s).

By default, Plug-in framework looks for the plug-ins under the `OIM_HOME/plugins` directory. Additional plug-in directories can also be specified.

If a monitoring thread is enabled, then the plug-in framework monitors all the additions, modification, and deletions of plug-in zip files under the registered plug-in directories in the file system, and automatically reloads the plug-ins. Plug-in metadata such as name, version, and ID is read from the plug-in zip and is maintained in memory. This metadata is updated based on any file changes. The latest plug-in zip file is considered to be the current version of the plug-in. For details about how to configure the file store, see "Configuring Plug-ins" on page 17-5.

Note: Oracle recommends not to use the file store in production. File store is more suitable during plug-in development because it is easy to change the plug-in, and you are required to change only the file in the file system. There is no need to register. However, in production, plug-ins are not changed often, and therefore, avoid using the file store because of certain disadvantages. It adds the overhead of file store monitoring. In addition, the plug-ins are required to be replicated in all nodes of a cluster for the clustered deployment of Oracle Identity Manager.

17.1.2.2 Database Store

Plug-ins can be stored in the Oracle Identity Manager database, so that they are accessible from any node in a cluster. The Plug-in Framework uses Operation DB as the database store. This type of store is appropriate for a production environment.

You must explicitly register any plug-ins that are stored in the database. You can use the Plugin Registration Utility, which is a command-line tool, to register and deregister plug-ins. You can also use the `registerPlugin` API for this purpose. See "Registering and Unregistering Plug-ins By Using APIs" on page 17-7 for more information about registering plug-ins.

Note: After registering a plug-in, the server must be restarted. However, restarting the server might also depend on the feature that defines the plug-in point.

17.2 Using Plug-ins in Deployments

As already mentioned in this document, plug-ins are used for customizing the default functionality in an Oracle Identity Manager deployment. The number of supported plug-in points is a defined and constrained set. Therefore, you can use the plug-in points to extend the functionality only for the list of supported plug-in points. See "Plug-in Points" on page 17-3 for a list of the supported plug-in points.

17.3 Plug-in Points

Table 17–1 lists the Java interfaces that act as plug-in points in Oracle Identity Manager:

Table 17–1 Plug-in Points

Plug-in Point	Description
oracle.iam.ldapsync.LDAPContainerMapper	This is used by LDAP synchronization to determine which user/role container should be used to create the user/role in LDAP.
oracle.iam.platform.kernel.spi.EventHandler	This is the kernel event handler. See Chapter 18, "Developing Event Handlers" for information about kernel event handlers.
oracle.iam.platform.auth.api.LoginMapper	This is an implementation of a LoginMapper maps the JAAS user principal name to the corresponding Oracle Identity Manager username. This plug-in point is used to override the default mapping of JAAS user principal name to Oracle Identity Manager username for SSO scenarios. The default implementation returns the same value as the JAAS user principal name. This plug-in point is typically used in SSO scenarios where the JAAS user principal name and the Oracle Identity Manager username might be different. For example, the SSO system might set the email as the JAAS username but no user with that username exist in Oracle Identity Manager. For Oracle Identity Manager to recognize that user, the JAAS user principal name must be mapped to the Oracle Identity Manager username. This can be done by implementing a plug-in for LoginMapper, as shown:

```
public class CustomLoginMapper implements LoginMapper{
public String getOIMUserID(String jaasPrincipal) throws
MappingException {
                return getUserName(jassPrincipal);
  }

private String getUserName(String emailID){
                String userName = null;

                //Use usermgmt APIs to get the username
corresponding to this email id
                return userName;
 }
}
```

oracle.iam.identity.usermgmt.api.PasswordVerifier	This is used for verification of old password while changing the user's password. The class that is to be used for this validation is configured in the OIM.OldPasswordValidator system property. By default, use the container based authentication for verifying old password.
oracle.iam.request.plugins.StatusChangeEvent	This allows running of custom code during request status change.
oracle.iam.request.plugins.RequestDataValidator	This is used for custom validation of request data after submission.
oracle.iam.request.plugins.PrePopulationAdapter	This is used to prepopulate an attribute value by running custom code during request creation.
oracle.iam.scheduler.vo.TaskSupport	This is used to run the job in context. Execute method of the task is retrieved through the plug-in and is loaded.
oracle.iam.identity.usermgmt.api.UserNamePolicy	This is an implementation of username policies that are used to generate/validate username.
oracle.iam.identity.usermgmt.api.ReservationInLDAP	This is an implementation for reservation of user attributes in LDAP.

17.4 Configuring Plug-ins

You use the oim-config.xml file in the MDS to configure the following:

> **See Also:** "Configuring the oim-config.xml File" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager* for information about configuring the oim-config.xml file

- The directory or directories in which the files store will look for plug-ins.
- Whether to activate a thread that monitors the file store for any changes; the thread checks the zip files or exploded files in all the plug-in directories.

  The monitoring thread is typically activated in a dynamic development environment since plug-ins are being added or modified in such an environment; it can be inactive in a production system which contains a set of plug-ins . This is tracked by the reloadingEnabled attribute.
- The time interval at which the monitoring thread wakes up and looks for any changes.

The following is a code snippet from the `oim-config.xml` file:

```
<pluginConfig storeType="common">

    <storeConfig reloadingEnabled="true"

      reloadingInterval="20">

      <!--

        Plugins present in the OIM_HOME/plugins directory are added by default.

        For adding more plugins, specify the plugin directory as below:

        <registeredDirs>/scratch/oimplugins</registeredDirs>

        <registeredDirs>/scratch/custom</registeredDirs>

      -->

    </storeConfig>

  </pluginConfig>
```

In this example:

- The `common` store designation tells the framework to monitor both database and file stores

  > **Note:** Do not modify the `Store` value; `common` is appropriate in all environments.

- One directory is configured; additional directories can be configured by simply adding more <registeredDirs> tags.
- The monitoring thread is active and looks for plug-in changes every 20 seconds by default.

Monitoring is typically active in development environments only. If you switch between active and inactive, you must restart the application server for the change to take effect.

Note: Restarting the application server is required for any changes made to plug-in data in the oim-config.xml file.

17.5 Developing Custom Plug-ins

This section describes how to develop custom plug-ins. It contains the following sections:

- Developing Plug-ins
- Declaring Plug-ins

17.5.1 Developing Plug-ins

To develop a plug-in:

1. Identify the plug-in point to extend.
2. Identify the Java class that implements the plug-in point interface. Package the Java class and other dependent classes into a JAR file. Put the JAR file in the lib/ directory.
3. Create the plugin.xml file. See "Declaring Plug-ins" on page 17-7 for details.
4. Identify the resource files required by the plug-in, such as property files, resource bundles, and image files.
5. Zip the entire package.

   An Oracle Identity Manager plug-in is distributed as a ZIP file with a specified directory structure. The directory structure is as follows:

   - **The plugin.xml file:** The XML file contains the metadata associated with all the plug-ins such as the plug-in point it extends, the class implementing the plug-in, name, and the version number. All the fields in the XML are mandatory except the name. If the name is not given, then plugin class name is used as the name.
   - **The lib/ directory:** The lib/ directory consists of JAR files that contains the classes implementing the plug-in logic and the dependent library JAR files. In most instances, this directory consists of a single JAR file with the implementation of all the plug-ins that are specified in plugin.xml.
   - **The resources/ directory:** Contains resource files required by the plug-in, such as property files, resource bundles, and image files. These resources given in the resources directory of the plug-in zip can be accessed as follows:

     ```
     this.getClass().getClassLoader().getResourceAsStream(<resource_name>);
     ```

   - **The META-INF/ directory:** Contains XML files showing plug-in points for event handlers. Some services, such as the notification service, read the XML files from MDS or from the META-INF/ directory of the plug-in.

   Multiple plug-ins implementing the same plug-in point can be part of the same ZIP file.

A plug-in has a Java class that implements the plug-in point interface. The plug-in library (JAR) can contain other dependent classes as well, but the class implementing the plug-in is the only one that is exposed to the feature. This class must be specified in plugin.xml.

6. Place the ZIP file in the file store (the *OIM_HOME*/plugins/ directory) or database store.
7. If the ZIP is placed in the database store, then register the plug-in by using the Plug-in Registration Utility, as described in "Registering Plug-ins" on page 17-7.

17.5.2 Declaring Plug-ins

To extend the functionality provided by Oracle Identity Manager, you can declare the plug-ins for the application.

A plug-in has a Java class that implements the plug-in point interface. Be sure to assign unique names to all the plug-ins associated with a specific plug-in point. If the plug-in names are non-unique, an exception will be thrown during plug-in registration.

Declare the plug-ins in the `plugin.xml` file. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<oimplugins>
....
<plugins pluginpoint="oracle.iam.sample.passwdmgmt.service.PasswordElement">
        <plugin pluginclass=
        "oracle.iam.sample.passwdmgmt.custom.NumCustomPasswordElement"
        version="1.0.1" name="num pwd element"/>
        <plugin pluginclass=
        "oracle.iam.sample.passwdmgmt.custom.DictionaryPasswordElement"
        version="1.0.1" name="Dictionary password element" />
</plugins>
....
</oimplugins>
```

Note: You can have multiple versions of the plug-in stored and the feature can request a specific version of the plug-in from the plug-in framework. By default, all of the current plug-in points load the latest version of the plug-ins.

The XML shows two plug-in declarations. Both the plug-ins extend from the same plug-in point.

17.6 Registering Plug-ins

You can register the plug-ins by using APIs and Plugin Registration Utility.

- Registering and Unregistering Plug-ins By Using APIs
- Registering and Unregistering Plug-ins By Using the Plugin Registration Utility

17.6.1 Registering and Unregistering Plug-ins By Using APIs

You can use the following APIs for registration-related tasks:

- PlatformService.registerPlugin

- PlatformService.unRegisterPlugin

Here is an example:

```
System.out.println("Creating client....");
String ctxFactory = "weblogic.jndi.WLInitialContextFactory";
String serverURL = "t3://OIM_HOSTNAME:OIM_PORT";
System.setProperty("java.security.auth.login.config",
"OIM_CLIENT_HOME/conf/authwl.conf");
String username = "USER_NAME";
char[] password = "PASSWORD".toCharArray();
Hashtable env = new Hashtable();
env.put(OIMClient.JAVA_NAMING_FACTORY_INITIAL,ctxFactory);
env.put(OIMClient.JAVA_NAMING_PROVIDER_URL, serverURL);

oimClient = new OIMClient(env);
System.out.println("Logging in");
oimClient.login(username, password);
PlatformService service = platform.getService(PlatformService.class);
File zipFile = new File(fileName);
FileInputStream fis = new FileInputStream(zipFile);
int size = (int) zipFile.length();
byte[] b = new byte[size];
int bytesRead = fis.read(b, 0, size);
while (bytesRead < size) {
bytesRead += fis.read(b, bytesRead, size - bytesRead);
}
fis.close();
service.registerPlugin(b);
service.unRegisterPlugin(pluginID, version);
```

Note: "Using OIMClient" on page 20-2 for information about using OIMClient for developing clients to integrate with Oracle Identity Manager.

17.6.2 Registering and Unregistering Plug-ins By Using the Plugin Registration Utility

You can use the Plugin Registration Utility for registering and unregistering plug-ins. The utility uses the following files:

- pluginregistration.xml
- ant.properties

These files are located in the *OIM_HOME*/plugin_utility/ directory.

Note: Plug-in registration utilities require Apache Ant version 1.7 or later.

Before using the utility, perform the following:

1. Set the values for wls.home and oim.home in ant.properties.

   For example:

   ```
   wls.home =.../middleware/wlserver_10.3
   oim.home =..../middleware/Oracle_IDM1/server
   ```

In addition, set the path for mw.home in the ant.properties file. Also, uncomment the following:

```
#login.config=${oim.home}/config/authwl.conf
```

2. Build the wlfullclient.jar in Oracle WebLogic server:
   a. Change directories to *WLS_HOME*/server/lib.
   b. Run the following command:

      ```
      java -jar ../../../modules/com.bea.core.jarbuilder_1.3.0.0.jar
      ```

Note: The exact JAR file version can be different based on the WLS. Use the corresponding file with the name as `com.bea.core.jarbuilder` at the *WLS_HOME*/../modules/ directory.

Registering a Plug-in

To register a plug-in:

1. Execute the ant target "register":

   ```
   ant -f  pluginregistration.xml register
   ```

2. This will prompt for the Oracle Identity Manager username and password along with the server information and the location of the plugin zip file. Enter the complete path of the zip file location.

Unregister a Plug-in

To unregister a plug-in:

1. Execute the ant target "unregister":

   ```
   ant -f  pluginregistration.xml unregister
   ```

2. This will prompt for the Oracle Identity Manager username and password along with the server information and the classname of the plug-in class. Enter the classname with the complete package.

Re-registering and Activating an Old Plug-in Version

To re-register and activate an older version of a plug-in:

1. Copy the old plug-in ZIP file in the *OIM_HOME*/plugins/ directory.
2. Execute the ant target "register":

   ```
   ant -f  pluginregistration.xml register
   ```

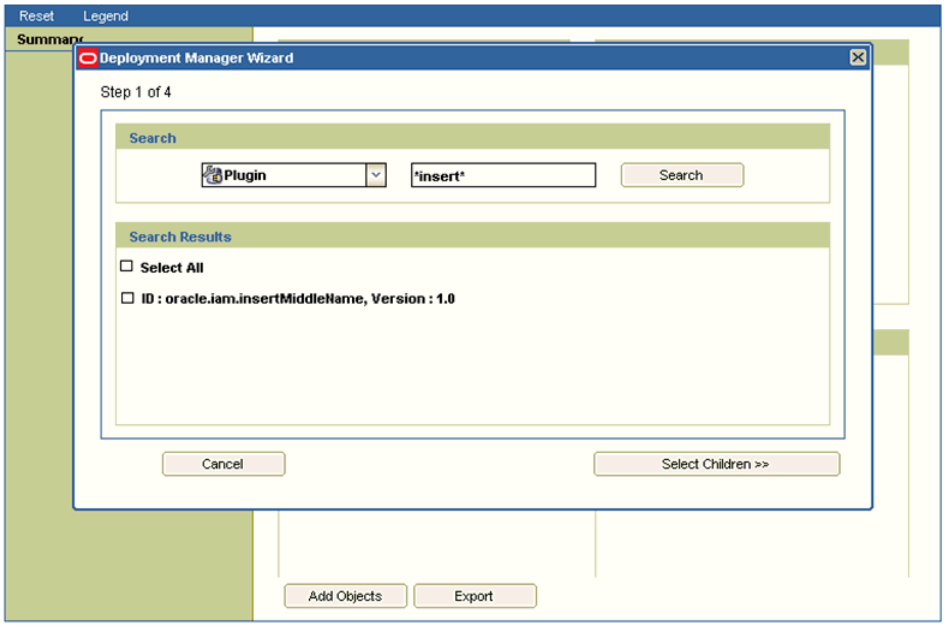
3. This will prompt for the Oracle Identity Manager username and password along with the server information and the location of the plug-in ZIP file. Enter the complete path of the ZIP file.

   A message is displayed stating that the plug-in is successfully re-registered.

17.7 Migrating Plug-ins

The Deployment Manager supports migrating plug-ins from one deployement of Oracle Identity Manager to another. For example, the event handlers can be implemented in a test environment, and then migrated to the production environment by using the Deployment Manager. Figure 17–2 shows exporting plug-ins via the Deployment Manager:

Figure 17–2 Exporting Plug-ins



See Also: "Migrating Incrementally Using the Deployment Manager" in *Administering Oracle Identity Manager* for information about the Deployment Manager

18 Developing Event Handlers

This chapter describes the concepts related to orchestration and how to write custom event handlers to extend the functionalities of Oracle Identity Manager. It contains the following topics:

18.1 Orchestration Concepts

In an Identity Management system, any action performed by a user or system is called an operation. Examples of operations are creating users, modifying roles, and creating password policies. The process of any Oracle Identity Manager operation that goes through a predefined set of stages and executes some business logic in each stage is called an *orchestration*. The type of object that is changed by the orchestration is called an orchestration target. The data that is required to carry out the orchestration operation is called orchestration parameter.

A bulk orchestration is the process of orchestrating same operation on multiple entities. For example, if you want to update the organization of multiple users, then you can submit a bulk orchestration. As a result, the operation on all the entities are performed in a single call.

Note: If custom event handlers are required to be introduced for lock/unlock operations, then you must implement bulk orchestrations. From the UI, bulk orchestrations are triggered for a single user lock/unlock operation.

Orchestration is divided into predefined steps called stages. Every operation moves through these stages until it reaches finalization. Orchestration has the following stages:

- **Validation:** Stage to perform validation on the orchestration, such as validity of orchestration parameters. Orchestration parameter is the data that is required to carry out the orchestration operation.
- **Preprocess:** Stage to perform orchestration parameter manipulations or get approvals or perform Segregation of Duties (SoD) checks.
- **Action:** Stage in which the action takes place.
- **Audit:** Stage in which the auditing of operation is performed.
- **Postprocess:** Stage in which consequent operations related to the current operation takes place. Examples of consequent operations are auto role membership and policy evaluation on a user creation.
- **Finalization:** Last stage in the process to perform any clean up.

Each operation performed can have consequences for users or other entities. For example, creating a user might result in provisioning of resources to that user, and creating a new password policy can make certain user passwords invalid and require changes during next login. Each consequence is represented as an orchestration. A differed consequence is executed before the finalization of the current orchestration. An immediate consequence is executed immediately after the current event handler returns, before proceeding to the next event handler on the current orchestration. You can customize the consequences of some operations, such as create, modify, delete, enable, disable, lock, and unlock users, by writing event handlers, as described in subsequent sections.

There are orchestrations for which the starting point is the postprocess stage. If you are reconciling users from a trusted source or bulk loading users and want to add this data as is in Oracle Identity Manager. When the data is in Oracle Identity Manager, you can perform postprocess operations on the users to compute autogroup membership or evaluate policies. Therefore, reconciliation engine or bulk load utility submits postprocess-only orchestrations.

An *event handler* is a piece of code that is registered with an orchestration on various stages. These event handlers are invoked when the relevant orchestration stage is performed. Event handlers can either be asynchronous or synchronous. A synchronous event handler returns with a response right away, whereas an asynchronous event handler completes at a later stage. An event handler can be conditional, which means that the event handler is executed when certain conditions are satisfied.

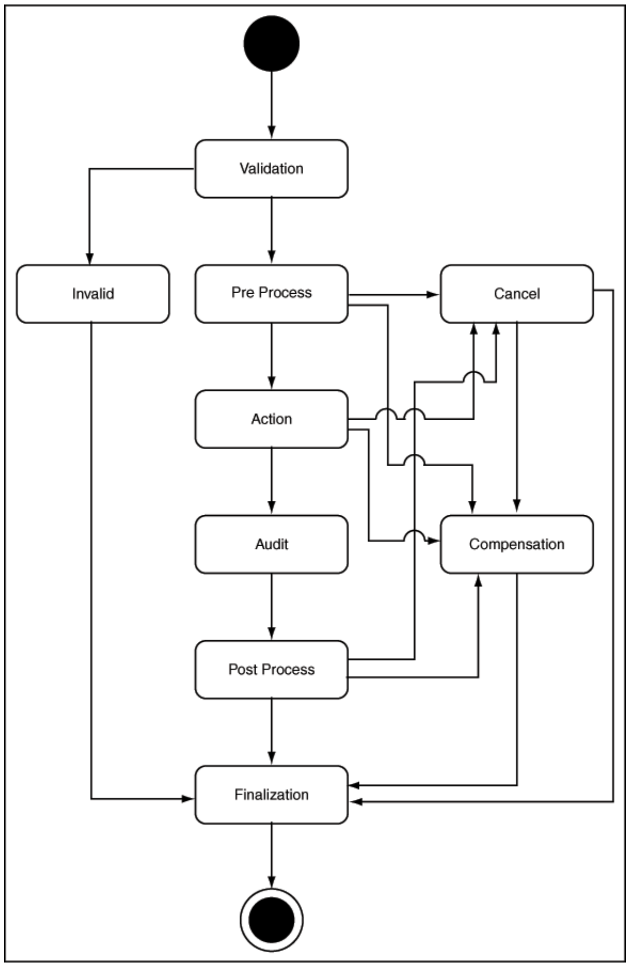
What happens at each stage of orchestration is determined by branching and by the event handler, if any, that is deployed at that stage. If a stage has a branch, responses from the event handlers decide which branch to take. If a stage has no event handlers, or event handlers respond with no recommendation, then the operation follows the default path and moves to the next stage. However, a process can move to some out-of-the-band stages if the event handlers are invalid or canceled. These stages are:

- **Invalid:** Process is moved to this stage if orchestration validation fails.
- **Veto:** Process is moved to this stage if any of the preprocess event handlers are vetoed. For example, if approvals are rejected by the approver, then orchestration is vetoed.
- **Cancel:** Process is moved to this stage if the operation is stopped by calling the cancel method.

- **Compensation:** Process is moved to this stage if the operation is rolled back by calling the compensate method.

Figure 18–1 shows the various orchestration stages:

Figure 18–1 Orchestration Stages



Note: Dynamic Monitoring Service (DMS) can be used to view performance metrics. The `OIM_Orchestration` DMS metric is present for monitoring orchestration performance. It shows the orchestration operations executed and the time taken to perform an orchestration operation.

18.2 Using Custom Event Handlers

Oracle Identity Manager allows you to implement Service Provider Interfaces (SPIs) to customize the functionality of orchestration operations. Only customization of preprocess, postprocess, validation, and finalization stages of an operation in an entity orchestration is supported.

The following are examples of event handler implementation:

- When a user is created, the account status (enabled or disabled) is to be set based on some rules. A preprocess event handler can be implemented to achieve this.

- Users of type Contractors must have an email address at the time of creation. Other users can be created without email address. A validation event handler can be used to validate if the user is a Contractor, and then allow or disallow the user creation based on the validation result.
- Users of type Agents are to be notified in the user's alternate email address after the users are created. This can be achieved by implementing a postprocess event handler.

Postprocess event handlers are most commonly implemented to meet business requirements. The following example describes how a postprocess event handler implementation can meet the given requirement:

Requirement

If the enterprise user is a Contractor, then after the user is created in Oracle Identity Manager, the user must be registered in the Contractor Registration System, which is an external application. This application is a database application. The database has a structure that stores the User ID, Contractor ID, First Name, and Last Name attributes of the users. After successful registration, the Contractor ID of the users must be retrieved and updated in the user's profile in Oracle Identity Manager.

Solution

This use case can be developed as a plug-in and deployed on Oracle Identity Manager. The plug-in can be used to retrieve the Contractor ID or any configured column name from specified database table and update the user profile in Oracle Identity Manager.

A postprocess event handler can be implemented and registered for the create operation of the user entity. It is a conditional event handler that executes for users only with type as Contractor. If the user type is Contractor, then the event handler connects to the external application to retrieve the Contractor ID based on the Oracle Identity Manager user ID, and update the user profile in Oracle Identity Manager with contractor ID.

The following is another common example of postprocess implementation of event handlers:

Custom attribute generation if the data that is reconciled into Oracle Identity Manager is not enough to implement all use cases and extra attributes need to be generated based on the reconciled data. This is a common use case, especially when the custom attributes are used in the role membership rules or access policies.

18.3 Orchestration Operations for Entities

Table 18–1 lists the orchestration operations supported for various entities.

Table 18–1 Orchestration Operations for Entities

Entity	Orchestration operation
User	CREATE MODIFY DELETE DISABLE ENABLE LOCK UNLOCK CHANGE_PASSWORD RESET_PASSWORD ADD_PROXY UPDATE_PROXY REMOVE_PROXY REMOVE_ALL_PROXIES GET_ALL_PROXIES SELFSETCHALLENGE EVALUATE_POLICIES
Organization	CREATE MODIFY ENABLE DISABLE DELETE
Role	CREATE MODIFY MODIFY_RULE DELETE
RoleCategory	CREATE MODIFY DELETE
RoleUser	CREATE MODIFY DELETE
RoleRole	CREATE MODIFY DELETE

18.4 Developing Custom Event Handlers

An event handler consists of the following:

- **Java code:** Implementation of the operations
- **XML definition:** Association with the relevant orchestration at the right stage

- **Plug-in definition:** Registration of the event handlers and any extension code with Oracle Identity Manager plug-in framework

Developing a custom event handler comprises of implementing the operation through Java code, writing the XML definition, and creating and registering a plug-in. These are described in the following sections:

- Implementing the SPI and Creating a JAR
- Defining Custom Events Definition XML
- Creating and Registering a Plug-in ZIP

18.4.1 Implementing the SPI and Creating a JAR

This section describes how to write the JAVA code by implementing the SPI, and thereafter, create a JAR file in the following sections:

- Development Considerations
- Methods and Arguments
- Code Samples
- Creating a JAR File With Custom Event Handler Code

18.4.1.1 Development Considerations

The following points must be considered for writing custom event handlers:

- The supported orchestration stages in which a custom event handler can be registered are validation, preprocess, and postprocess.
- Validation, preprocess, and postprocess event handlers can be conditional. This means that the event handler will execute only if a particular condition is met.

  You can make the event handler conditional by implementing the oracle.iam.platform.kernel.spi.ConditionalEventHandler interface and its isApplicable method. Context data and orchestration parameters are available in this method. For conditional event handlers, the applicability of event handlers is computed when the operation is initiated. Therefore, if a context or orchestration parameters are modified during the orchestration flow, then it might lead to execution of event handlers that must not be executed.
- The event handlers can handle single as well as bulk entities.
- The event handlers can have associated failure handlers that callbacks certain operations on the parent handlers.
- Because retry of event handlers is supported, the event handlers can be re-entrant.
- When reconciliation submits postprocess orchestrations, it submits bulk orchestrations. The bulkExecute method on the event handlers is called for these orchestrations. Therefore, make sure to implement this method.
- If data is to be passed between custom event handlers, you can pass it by using inter event data. Calling the getInterEventData() method on orchestration returns a hashmap. In this map, you can put any object with key beginning with custom, and you can access this data in subsequent custom handlers. Do not modify or delete any predefined inter event data that is part of the same hashmap.
- To make API calls inside event handlers for write or delete operations, get the API services by using Platform.getServiceForEventHandlers method. API calls that are

made using the services obtained through this method are performed synchronously including the postprocessing.

- Return type of event handlers, except validation handlers, are shown in the following table:

Event Handler Type	On Success	On Failure
Synchronous	new EventResult() in the execute method and new BulkEventResult() in bulk version of the execute method	EventFailedException
Asynchronous	Return null	EventFailedException

- You must not define object-level variables at the event handler.

18.4.1.2 Methods and Arguments

Table 18–2 lists the methods that you can implement in the various orchestration stages:

Table 18–2 Methods to Implement Event Handlers

Method	Applicable Orchestration Stage	Description
initialize	preprocess, postprocess	This method is used to open connections and pool state or resources.
execute for single entity	preprocess, postprocess	This method is used to read the input attributes of the underlying operation and update to different values, if required.
execute for bulk orchestration	preprocess, postprocess	This method is used to read the input attributes of multiple underlying operations and update to different values, if required.
isApplicable	conditional	This method is used in conditional handlers to determine if the prerequisite condition for the event handler execution is met.
validate	validation	This method is used for validation handlers to validate input data.
cancel	preprocess, postprocess	This method is called when the orchestration operation is canceled.
compensate	preprocess, postprocess	This method is called when the orchestration operation is compensated.

For methods, such as execute, the following argument values are available:

- IDs that you can include in the code for troubleshooting purpose, which includes:
  - **Process ID:** The ID of the orchestration instance
  - **Event ID:** The ID of the event handler instance
- Orchestration object that consists of details of the underlying entity instance. This consists of:
  - Maps (key value pairs) containing *ENTITY_ATTRIBUTE*, *VALUE* from which the input attributes of the underlying entity is read.

– Entity ID: To update back the values for the same or a different entity, use Entity Manager API and pass the Entity ID and data to it. For bulk orchestration, you get multiple Entity IDs and Maps.

Note: Use Platform.getServiceForEventHandlers to get the services for calling create, update, and delete operations in event handlers.

18.4.1.3 Code Samples

This section provides code samples that illustrate how to write various kinds of event handlers.

Example 1: Custom Email Validation

Example 18–1 shows a sample custom validation handler code fragment that checks to ensure that the ampersand character (@) is used in the email id of the user.

Example 18–1 Custom Email Validation

```
public void validate(long processId, long eventId, Orchestration orchestration) throws
ValidationException, ValidationFailedException {
    HashMap<String, Serializable> parameters = orchestration.getParameters();
    String email = (parameters.get("Email") instanceof ContextAware) ? (String) ((ContextAware)
parameters
                .get("Email")).getObjectValue() : (String) parameters
                .get("Email");
        if (!(email.contains("@"))) {
             throw new ValidationFailedException("Email doesn't contain @");
         }
    }
```

Example 2: Custom Preprocess Event Handler to Set Middle Name

Example 18–2 shows a sample custom preprocess event handler code fragment that sets the middle name to the first letter of the first name if the a value is not provided for middle name.

Example 18–2 Custom Preprocess Event Handler to Set Middle Name

```
// the middle initial when the user doesn't have a middle name
    public EventResult execute(long processId, long eventId,
            Orchestration orchestration) {
        HashMap<String, Serializable> parameters = orchestration
                .getParameters();
        // If the middle name is empty set the first letter of the first name
        // as the middle initial
        String middleName = getParamaterValue(parameters, "Middle Name");
        if ((middleName == null) || middleName.equals("")) {
            String firstName = getParamaterValue(parameters, "First Name");
            middleName = firstName.substring(0, 1);
            orchestration.addParameter("Middle Name", middleName);
        }
        return new EventResult();
    }

    private String getParamaterValue(HashMap<String, Serializable> parameters,
            String key) {
            if(parameters.containsKey(key)){
```

```
    String value = (parameters.get(key) instanceof ContextAware) ? (String) ((ContextAware)
parameters
            .get(key)).getObjectValue() : (String) parameters.get(key);
    return value;
            }
            else{
                return null;
            }
    }
```

Example 3: Custom Post-process Event Handler to Provision Resource Object

Example 18–3 shows a sample custom post process event handler code fragment that provisions a resource object OBJ005 to a user whose role is ROLE 005:

Example 18–3 Sample Custom Post Process Event Handler

```
// This custom post process event handler provisions resource object 'OBJ005'
// to a user who has role 'ROLE 005'
public EventResult execute(long processId, long eventId,
  Orchestration orchestration) {
    tcUserOperationsIntf userOperationsService =
    Platform.getService(tcUserOperationsIntf.class);
try {
  String userKey = getUserKey(processId, orchestration);
  if (hasRole(userKey, "ROLE 005")) {
     long objKey = findObject("OBJ001");
userOperationsService.provisionResource(Long.getLong(userKey), objKey);
}
} catch (Exception e) {
throw new EventFailedException(null, "Error occurred", null, null, e);
}

return new EventResult();
}

// This method retrieves the key of the user entity on which an operation
// is performed
// This method shows how to retrieve the operation being performed, entity type
// and the associated value objects
private String getUserKey (long processID, Orchestration orchestration) {
  String userKey;
  String entityType = orchestration.getTarget().getType();
  EventResult result = new EventResult();

if (!orchestration.getOperation().equals("CREATE")) {
userKey = orchestration.getTarget().getEntityId();
} else {
OrchestrationEngine orchEngine = Platform.getService(OrchestrationEngine.class);
userKey = (String) orchEngine.getActionResult(processID);
}
return userKey;
}

// This method checks if a given user has a given role.
// It demonstrates how to invoke a OIM 11g API from a custom event handler
private boolean hasRole(String userKey, String roleName)
  throws Exception {
  RoleManager roleManager = Platform.getService(RoleManager.class);
  List<Role> roles = roleManager.getUserMemberships(userKey, true);
  for (Iterator iterator = roles.iterator(); iterator.hasNext();) {
Role role = (Role) iterator.next();
if (roleName.equals((String)role.getAttribute("Role Name"))) {
return true;
}

}
return false;
}

// This method finds details about a resource object with the given name.
// It demonstrates how to invoke a 9.1.x API from a custom event handler
private long findObject(String objName) throws Exception {
  long objKey = 0;
  tcObjectOperationsIntf objectOperationsService =
  Platform.getService(tcObjectOperationsIntf.class);
HashMap params = new HashMap();
params.put("Objects.Name", objName);
tcResultSet objects = objectOperationsService.findObjects(params);
for (int i = 0; i < objects.getRowCount(); i++) {
  objects.goToRow(i);
  if (objects.getStringValue("Objects.Name").equals(objName)) {
  objKey = objects.getLongValue("Objects.Key");
}
}
 return objKey;
}
```

Example 4: Custom User Postprocess Event Handler With bulkExecute Method

Example 18–4 shows how to loop through users that are part of a bulk user create orchestration.

Example 18–4 Custom User Postprocess Event Handler With bulkExecute Method

```
public BulkEventResult execute(long processId, long eventId, BulkOrchestration
orchestration){

HashMap<String, Serializable>[] orchParamArray =
orchestration.getBulkParameters();

       // Array of user keys
        String [] entityIds = orchestration.getTarget().getAllEntityId();
        for(int i=0; i< entityIds.length; i++){
        }

}
```

Example 5: Using Context in isApplicable method

Any operation in Oracle Identity Manager can take place in more than one context. For example, creating a user can happen in four different contexts, which are administrator creating a user as a direct operation, administrator creating a user by raising a request, creating a user through self registration, and user creation through trusted source reconciliation. In all these scenarios, Oracle Identity Manager submits the same user creation orchestrations having the same parameter names and values with same data types.

Example 18–5 shows how to find the context in which this operation is performed to figure out the applicability of the event handler.

Example 18–5 Using Context in the isApplicable Method

```
public boolean isApplicable(AbstractGenericOrchestration orchestration) {    //
Request Context
    if (ContextManager.getContextType() == ContextTypes.REQUEST) {
    }
    // Recon context
    if (ContextManager.getContextType() == ContextTypes.RECON) {
    }



}
```

18.4.1.4 Creating a JAR File With Custom Event Handler Code

To create a JAR with custom event handler code:

1. Implement one of the SPIs mentioned in Table 18–3 to write a custom preprocess, postprocess, or validation handler.

Table 18–3 SPIs to Write Custom Event Handlers

Stage	SPI to implement
Preprocess	oracle.iam.platform.kernel.spi.PreProcessHandler
Postprocess	oracle.iam.platform.kernel.spi.PostProcessHandler
Validation	oracle.iam.platform.kernel.spi.ValidationHandler
Finalization	oracle.iam.platform.kernel.spi.FinalizationHandler

> **See Also:** See *Oracle Fusion Middleware Java API Reference for Oracle Identity Manager* for information about the SPIs listed in Table 18–3

2. Include the following JAR files in the class path to compile a custom class:

   From the *OIM_ORACLE_HOME*/server/platform/ directory:

   - iam-platform-kernel.jar
   - iam-platform-utils.jar
   - iam-platform-context.jar
   - iam-plaftorm-authz-service.jar

   From the *OIM_ORACLE_HOME*/designconsole/lib/ directory:

   - oimclient.jar
   - xlAPI.jar

   If some other Oracle Identity Manager JAR files are required for compilation, then these can be found in the directories mentioned in this step.

3. Create a JAR file of the custom class.

18.4.1.5 Handling Exceptions

For event handler exception handling, you must use conventional JAVA exception handling methods. The following guidelines can be used for dealing with failures:

- In the event handler code, throw EventFailedException with the right arguments to indicate failure.
- Failures can be handled by registering failure handlers. As part of failure handler, you can implement necessary logic to remediate the failure. The failure handlers must return FailedEventResult with the following options as Response:
  - **CANCEL:** Indicates that operation must get canceled. The Cancel method on all event handlers that are executed and completed so far is called by Kernel in reverse order of execution.
  - **COMPENSATE:** Indicates that operation must get rolled back. The Compensate method on all event handlers that are executed and completed so far is called by Kernel in reverse order of execution.
  - **MANUAL_COMPLETE:** Indicates that the handler that failed is manually completed and will proceed with the rest of the event handlers.
  - **RETRY:** Indicates to kernel that the event handler that failed must be retried.
  - **NULL:** Indicates that there is no response or recommendation by the failed handler.

18.4.1.6 Managing Transactions

In the event handler XML file, set the tx attribute to true. If any exception is thrown in the event handler, then the transaction will be rolled back or committed.

18.4.2 Defining Custom Events Definition XML

The custom events definition XML is described in the following sections:

- Elements in the Event Handler XML Files
- Sample Event Definitions

18.4.2.1 Elements in the Event Handler XML Files

This section describes some of the elements and element attributes within Event Handlers XML files. It also describes a mandatory namespace for the event handler XML definitions.

Elements

The top-level (or parent) element in Event Handlers XML files is `eventhandlers`. Table 18–4 lists and describes sub-elements that are typically defined within the `eventhandlers` parent element.

Table 18–4 Typical Sub-elements within the eventhandlers Element

Sub-element	Description
`validation-handler`	Identifies the validations that will be performed on the orchestration.
`action-handler`	Identifies the operations that will be performed at preprocess, postprocess, and action stages.
`failed-handler`	Identifies the event handlers that will be executed if an event handler in the default flow fails.
`finalization-handler`	Identifies the event handlers to execute at the end of the orchestration. Finalization is the last stage of any orchestration.

Table 18–4 (Cont.) Typical Sub-elements within the eventhandlers Element

Sub-element	Description
`change-failed`	Identifies event handlers to be executed in parent orchestration upon consequence orchestration failures.
`out-of-band-handler`	Defines the event handlers for out-of-band orchestration flows, such as veto and cancel.
`compensate-handler`	Identifies the event handlers that will be executed in the compensation flow of the orchestration.

Element Attributes

The elements within event handlers XML files contain attributes. Table 18–5 lists and describes attributes that are typically defined within elements.

Table 18–5 Typical Attributes of Sub-elements within the eventhandlers Element

Element Attribute	Description
Name	The name of the event handler.
class	Full package name of the Java class that implements the event handler.
entity-type	Identifies the type of entity the event handler is executed on. A value of `ANY` sets the event handler to execute on any entity. Most commonly defined entity types are user, role, rolerole (role hierarchy), and roleuser (user role membership).
operation	Identifies the type of operation the event handler is executed on. A value of `ANY` sets the event handler to execute on any operation. Typical operations are create, modify, and delete.
order	Identifies the order (or sequence) in which the event handler is executed. Order value is in the scope of entity, operation, and stage. Order value for each event handler in this scope must be unique. If there is a conflict, then the order in which these conflicted event handlers are executed is arbitrary. Supported values are `FIRST` (same as Integer.MIN_VALUE), `LAST` (same as Integer.MAX_VALUE), or a numeral.
orch-target	Identifies the type of orchestration, such as entity orchestration, Toplink orchestration, and so on. The following is a list of supported values: ■ `oracle.iam.platform.kernel.vo.EntityOrchestration` ■ `oracle.iam.platform.kernel.vo.MDSOrchestration` ■ `oracle.iam.platform.kernel.vo.RelationOrchestration` ■ `oracle.iam.platform.kernel.vo.ToplinkOrchestration` The default value is `oracle.iam.platform.kernel.vo.EntityOrchestration`. This is the only supported type for writing custom event handlers.
sync	This attribute is operational in only the action-handler and change-failed elements. The sync attribute indicates whether the event handler is synchronous or asynchronous. Supported values are `TRUE` or `FALSE`. If set to `TRUE` (synchronous), then the kernel expects the event handler to return an EventResult. If set to `FALSE` (asynchronous), then you must return null as the event result and notify the kernel about the event result later. **Note**: The sync attribute must be set to `TRUE` for validation-handler elements.

Table 18–5 (Cont.) Typical Attributes of Sub-elements within the eventhandlers Element

Element Attribute	Description
stage	This attribute is operational in only the out-of-band-handler, action-handler, and failed-handler elements. The stage attribute indicates the stage at which the event handler is executed. The following is a list of supported values: ■ preprocess ■ action ■ audit ■ postprocess ■ veto ■ canceled
tx	This attribute is operational in only the out-of-band-handler, action-handler, compensate-handler, and finalization-handler elements. The tx attribute indicates whether or not the event handler should run in its own transaction. Supported values are TRUE or FALSE. By default, the value is FALSE.

Namespace Requirement in <eventhandlers> Element

All the event handler definitions must have the following mandatory namespace definition:

```
<eventhandlers xmlns="http://www.oracle.com/schema/oim/platform/kernel"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.oracle.com/schema/oim/platform/kernel
 orchestration-handlers.xsd">
```

18.4.2.2 Sample Event Definitions

Create a metadata XML file containing definitions of all the custom events, as shown in Table 18–6:

Example 18–6 Sample Metadata XML File for Custom Event Definitions

```
<?xml version='1.0' encoding='utf-8'?>
    <eventhandlers xmlns="http://www.oracle.com/schema/oim/platform/kernel"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.oracle.com/schema/oim/platform/kernel
     orchestration-handlers.xsd">

  <!-- Custom preprocess event handlers -->
  <action-handler
    class="oracle.oim.extensions.preprocess.SamplePreprocessExtension"
    entity-type="User"
    operation="CREATE"
    name="SetUserMiddleName"
    stage="preprocess"
    order="1000"
    sync="TRUE"/>

  <!-- Custom postprocess event handlers -->
  <action-handler
    class="oracle.oim.extensions.postprocess.SamplePostprocessExtension"
    entity-type="User"
    operation="CREATE"
    name="SamplePostprocessExtension"
    stage="postprocess"
    order="1000"
    sync="TRUE"/>

  <action-handler
    class="oracle.oim.extensions.postprocess.SamplePostprocessExtension"
    entity-type="User"
    operation="MODIFY"
    name="CustomResourceProv"
    stage="postprocess"
    order="1000"
    sync="TRUE"/>

  <!-- Custom validation event handlers -->
   <validation-handler
    class="oracle.oim.extensions.validation.SampleValidationExtension"
    entity-type="User"
    operation="CREATE"
    name="ValidateUserEmail"
    order="1000"/>
</eventhandlers>
```

18.4.3 Creating and Registering a Plug-in ZIP

To create plug-ins containing custom event handlers, you need to develop the appropriate event handler classes. See "Developing Plug-ins" on page 17-1 for detailed information about plug-ins and plug-in points.

To create a plug-in ZIP and register it:

1. Define the plug-in XML with the event handler plug-in point.

    Note: Ensure that plug-in point used in the plug-in definition is set to oracle.iam.platform.kernel.spi.EventHandler.

    The following is an example of a plug-in XML file:

```
<?xml version="1.0" encoding="UTF-8"?>
<oimplugins>
  <plugins pluginpoint="oracle.iam.platform.kernel.spi.EventHandler">
    <plugin pluginclass=
       "oracle.oim.extensions.preprocess.SamplePreprocessExtension"
        version="1.0"
        name="SamplePreprocessExtension">
    </plugin>
    <plugin pluginclass=
        "oracle.oim.extensions.postprocess.SamplePostprocessExtension"
         version="1.0"
         name="SamplePostprocessExtension">
    </plugin>
    <plugin pluginclass=
       "oracle.oim.extensions.validation.SampleValidationExtension"
        version="1.0"
        name="SampleValidationExtension">
    </plugin>
  </plugins>
</oimplugins>
```

2. Package the plug-in XML and the JAR file that contains the custom class or classes into a plug-in ZIP file.
3. Package the event handler XML that is defined using the information described in "Defining Custom Events Definition XML" on page 18-12 into the same zip in a directory called META-INF.
4. Register the plug-in by using plug-in registration utilities. See "Registering Plug-ins" on page 17-7 for additional information.

18.5 Sequencing the Execution of Event Handlers

The list of custom event handlers that you deployed and registered can be viewed by using Oracle Enterprise Manager. The event handlers are displayed in the order of invocation. Using this list of event handlers, you can sequence the order of execution of the event handlers.

To specify the order for any custom event handler, you must know the list of existing event handlers and their order for a given operation. To do so, you must invoke a mbean from the Enterprise Manager by performing the following steps:

1. Login to the Enterprise Manager.
2. On the left navigation pane, expand Weblogic Domain, and select OIM DOMAIN.
3. Right-click the domain name, and select **System Mbean Browser**.
4. Under Application Defined Mbeans, expand oracle.iam.
5. Navigate to ***OIM_SERVER_NAME***, **oim**, **Kernel**, and then click **OrchestrationEngine**.
6. Click the **Operations** tab.
7. Click the findEventHandlers method.
8. Provide entity name and operation name, and then click **Invoke**. The parameter values are case-sensitive. The possible parameter values are:
   - entity name: Values can be User, Role, or RoleUser
   - operation: Values can be CREATE, MODIFY, or DELETE

18.6 Writing Custom Validation Event Handlers

An approver can update the attribute values before approving a request. To ensure sanitization of the data entered by the approver, Oracle Identity Manager invokes validation handlers again when approver updates the request. This means that validation handlers configured for a particular entity and operations are invoked multiple times in a single request flow, when the request is submitted and when the approver modifies the request during approval workflow.

For example, when a self-registration request is submitted, the set of validation handlers configured for USER CREATE is run. Next, when the approver modifies the request to populate Organization or other user attributes, these validation handlers are re-run.

Therefore, custom validation handlers must be developed in such a way that the validation logic is re-entrant because they are invoked multiple times in single request flow.

Note: You can add a custom password validation for cases that are not available through Oracle Identity Manager password policies. For example, you can add a password validation to ensure that the password is not a word in a dictionary.

To add a custom password validation, add a custom validation event handler and set the operation to CHANGEACCOUNTPASSWORD for the event handler. Then, you can organize the order in which Oracle Identity Manager triggers the custom event handler.

Consider the following example use case:

There is a requirement of generating the HR Employee Number UDF by appending a random number to the value of the Department Number field. When the create user request or self-registration request is submitted, the HR Employee Number UDF will be auto-generated based on custom logic. If the approver edits the request during approval and modifies the Department Number value, then the HR Employee Number UDF should be re-calculated by using the new value provided for Department Number. But, if the approver does not change Department Number, then the previous values generated at the time of request submission should be used.

For this, a new validation handler must be developed for generating the HR Employee Number UDF by appending Department Number and a random number. This logic cannot be written in preprocess handler because preprocess handlers are invoked only once in the lifecycle of a request. The logic in this validation handler is as shown:

```
package custom.handlers;

import java.io.Serializable;
import java.util.HashMap;
import java.util.Random;

import oracle.iam.identity.usermgmt.api.UserManagerConstants;
import oracle.iam.identity.utils.Utils;
import oracle.iam.platform.kernel.ValidationException;
import oracle.iam.platform.kernel.ValidationFailedException;
import oracle.iam.platform.kernel.spi.ValidationHandler;
import oracle.iam.platform.kernel.vo.BulkOrchestration;
import oracle.iam.platform.kernel.vo.Orchestration;

public class EmployeeNumberGenerationHandler implements ValidationHandler {

@Override
public void initialize(HashMap<String, String> parameters) {

}

@Override
public void validate(long processId, long eventId, Orchestration orchestration)
throws ValidationException, ValidationFailedException {

HashMap<String, Serializable> contextParams = orchestration.getParameters();
//1. Generate UDF Employee number during request submission as Department Number
and a random number
//2. If request is in approval stage, then control has come here since approver
has modified the request
//2a: Check if approver has modified Department Number. If yes, then re-generate
if( !Utils.isRequestInApprovalStage()) //Utility method to find if request is in
approval stage or not? If it returns true, it means that approver is attempting to
update the request during approval
{

//Step 1:
String dept =
contextParams.get(UserManagerConstants.AttributeName.DEPARTMENT_NUMBER.getId()).to
String();
String en = dept+"_"+random();
contextParams.put("SSN", en);

}
else
{
String dept =
contextParams.get(UserManagerConstants.AttributeName.DEPARTMENT_NUMBER.getId()).to
String();
//compare with department number with which request was submitted, if modified by
approver; the regenerate SSN
if( Utils.isAttributeModifiedByApprover(orchestration ,
UserManagerConstants.AttributeName.DEPARTMENT_NUMBER.getId()) )

// //Utility method to find if approver has edited the particular attribute or not
, during approval?
{
String en = dept+"_"+random();
contextParams.put("SSN", en);
}

}

}

private String random() {
Random random = new Random();
String randomStr = "" + random.nextLong();
randomStr = randomStr.replaceAll("-", "");
return randomStr;
}


@Override
public void validate(long processId, long eventId,
BulkOrchestration orchestration) throws ValidationException,
ValidationFailedException {

}

}
```

18.7 Best Practices

As a best practice, analyze the operation before developing and implementing an event handler. If plug-in is supported for the operation, then use the plug-in for customization rather than developing an event handler. For example, username generation must be implemented by using the available plug-in, and do not attempt writing that as an event handler in the create user orchestration.

For information about points to consider for developing event handlers, see "Development Considerations" on page 18-6.

18.8 Migrating Event Handlers

To migrate event handlers from one deployment to another, you can export the event handlers in one deployment and then import them in another deployment by using the Deployment Manager. See "Migrating Incrementally Using the Deployment Manager" in *Administering Oracle Identity Manager* for information exporting and importing migration artifacts by using the Deployment Manager.

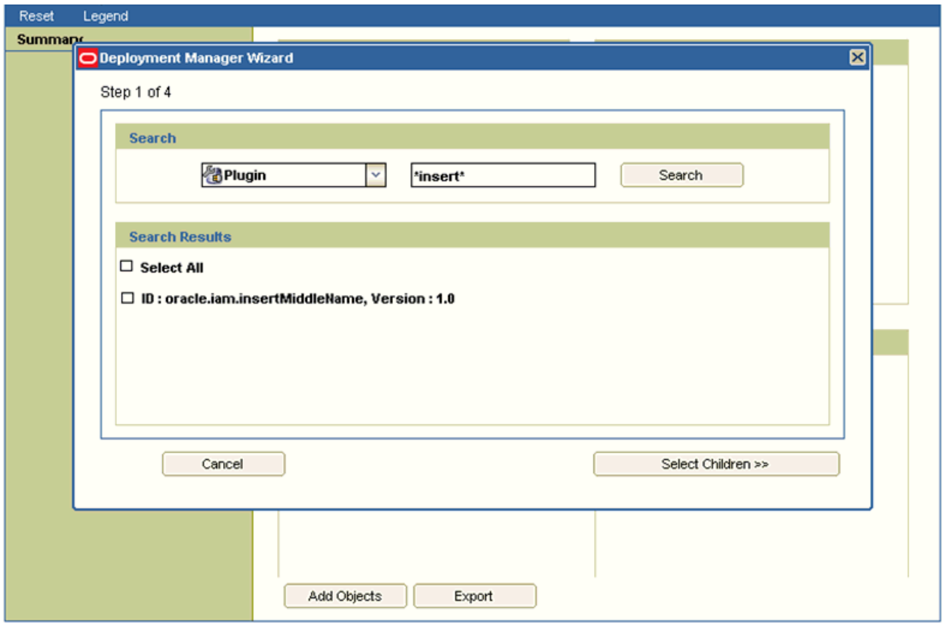
The Deployment Manager supports exporting and importing of default event handlers and custom event handlers. The following table lists the entities to be selected in the Deployment Manager for default and custom event handlers.

Event Handler Type	Deployment Manager Entity Selection
Default event handler	Orchestration event handler
Custom event handler	Plugin **Note:** In addition to custom event handlers, custom scheduled tasks and custom notification resolvers are also migrated as part of plug-ins. Therefore, you must select the plug-in with the custom event handler while exporting and importing.

The Deployment Manager supports migrating plug-ins, and the registered event handlers with the plug-ins, from one deployment of Oracle Identity Manager to another. See "Registering Plug-ins" on page 17-7 for information about registering and unregistering plug-ins by using the plug-in registration utility.

Figure 18–2 shows exporting plug-ins via the Deployment Manager.

Figure 18–2 Exporting Plug-ins



The exported plug-in is complete in itself and contains both the definition and the implementation. Therefore, you do not need to import/export the corresponding definition using other Deployment Manager entities, such as orchestration event handler in case of custom event handlers.

18.9 Troubleshooting Event Handlers

Table 18–6 lists common problems and causes or solutions to help troubleshoot your event handler if it is not triggered when the operation is executed.

Table 18–6 Troubleshooting Event Handlers

Problem	Cause/Solution
When a user is created through reconciliation, the custom preprocess event handlers are not triggered.	Reconciliation submits postprocess only orchestration where starting stage is postprocess.
When a user is created through reconciliation, the custom postprocess event handler is triggered but the logic inside the execute method is not triggered.	Reconciliation submits bulk orchestrations. Therefore, make sure to implement the bulkExecute method.
The orchestration operation taking too long to complete.	To determine the time spent on each event handler: **1.** Connect to http://*OIM_HOST*:*OIM_PORT*/dms/Spy as the WebLogic administrator. **2.** In the Metric Tables, click **OIM_EntityType_*ENTITY_NAME***, for example, **OIM_EntityType_User** for the user entity. How long each event handler is taking is displayed in the execute/bulkExecute column for orchestration/bulkOrchestration respectively.

Part VI

Customization

This part describes how to customize the user interfaces available with Oracle Identity Manager.

It contains the following chapter:

19 Customizing the Interface

This chapter explains how to customize various aspects of the user interfaces available in Oracle Identity Manager.

Note: Oracle Identity Manager 11*g* Release 2 (11.1.2.3.0) includes a number of UI pages based on earlier UI technologies known as transitional UIs. Due to technical differences, the transitional UIs are displayed in popup windows and have a different look and feel. These UIs are discussed in the relevant sections in this chapter.

This chapter describes customizing various aspects of the UI in the following sections:

- Managing Sandboxes
- Skin Customization in Oracle Identity Manager
- Customizing Pages at Runtime
- Securing a Task Flow Region Using EL Expressions
- Customizing Oracle Identity Manager Help
- Customizing the Home Page
- Developing Managed Beans and Task Flows
- Configuring Additional Request Form
- Migrating UI Customizations
- UI Customization Best Practices

19.1 Managing Sandboxes

All customizations and form management are performed in a sandbox. A sandbox allows you to isolate and experiment with customizations without affecting the environment of other users. Any user-interface changes made to a sandbox are visible only in the sandbox. You must create and activate a sandbox to begin using the customization and form management features. After customizations and extending forms are complete, you can publish the sandbox to make the customizations available to other users.

Some of the sandbox operations are:

- **Activate:** You must activate a sandbox to use it. After you activate the sandbox, any changes to UI metadata objects, for example pages and forms, are stored only in the sandbox. There can be only one active sandbox at a time. The information about the active sandbox is stored in the session. Therefore, a sandbox must be activated to continue with customization after every login to Oracle Identity Manager.

- **Deactivate:** Reverse operation to activating a sandbox. If no sandbox is active, then changes to metadata objects are not allowed, and therefore, no UI customization is allowed.
- **Publish:** You must publish a sandbox to merge the changes stored in the sandbox to the mainline and make it available to other users. After you publish the sandbox, the changes are merged to the mainline and cannot be reverted. The sandbox can no longer be activated, deactivated, exported, or deleted.

  Note: Before publishing a sandbox, export the sandbox to a ZIP file to have a backup of UI customizations done.

  Oracle recommends creating a backup of the MDS before publishing any sandbox. MDS backup can be created by using tools, such as Oracle Enterprise Manager. See "Creating MDS Backup" on page 24-2 for information about creating a backup of the MDS by using Oracle Enterprise Manager.

- **Publish in bulk and sequence:** If you have stored different types of changes in multiple sandboxes, then you can publish more than one or all the sandboxes in bulk to merge all the changes to the mainline and make then available to other users. While publishing the sandboxes in bulk, you can specify the sequence in which the sandboxes are to be published. See "Publishing Sandboxes in Bulk and Sequence" on page 19-8 for the procedure to publish sandboxes in bulk and sequence.

  Note: The bulk publish of sandboxes in sequence capability is available only after you have applied the Bundle Patch 11.1.2.3.170418. For instructions on downloading and applying the 11.1.2.3.170418 bundle patch, refer to the bundle patch documentation.

- **Export:** You can export all changes stored in the sandbox including sandbox metadata to a ZIP file. Then, you can import these changes to the same or another environment.
- **Import:** You can import the sandbox archive (ZIP file) to an environment. Imported sandbox can be used normally as it would have been created in the environment. Beware when importing sandboxes that any available sandbox with the same name will be overwritten by the imported sandbox.

  Caution: Any available sandbox with the same name is overwritten by the imported sandbox.

Sandbox management and sandbox operations resemble operations with concurrent versioning system. You can think of a sandbox as a branch in the versioning system. Creating a sandbox is similar to creating a branch. Activating a sandbox is similar to performing changes on top of the branch, and publishing a sandbox is similar to merging the content of the branch to the main branch, sometimes referred to as trunk.

Note: When you create a sandbox, a new branch is created. You can modify MDS content within that branch. Note that you will not be able to view the changes made in other sandboxes that are created later and published to the main branch. Similarly, when you try to merge this sandbox, a concurrent modification exception is generated. It is recommended that you edit the contents of the sandbox manually to remove the conflicting files. However, if manual editing is not possible, then create a new sandbox again and redo the change.

This section describes how to manage sandboxes in the following sections:

- Handling Concurrency Conflicts
- Creating a Sandbox
- Activating and Deactivating a Sandbox
- Viewing and Modifying Sandbox Details
- Exporting and Importing a Sandbox
- Publishing a Sandbox
- Publishing Sandboxes in Bulk and Sequence
- Deleting a Sandbox
- Reverting Changes to Default Settings

19.1.1 Handling Concurrency Conflicts

Multiple users can customize an application by using sandboxes. While doing so, the following types of concurrency conflicts might take place:

- **Conflicts within a sandbox:** Users overwriting changes created by other users, either directly by changing the same artifact, or indirectly by affecting files that are shared between the artifacts.

  Conflicts within a sandbox can arise when multiple users are customizing an application by using the same sandboxes at the same time, because more than one user may be attempting to customize the same artifact, or performing a customization task that indirectly affects other shared files. An example of a direct conflict is when different users attempt to customize the same page, the same fragment, or the same metadata file in the same layer. An example of an indirect conflict is when two users, each creating their own object, cause a conflict in the metadata file that tracks which new objects have been created by both saving their changes around the same time. Conflicts may also arise when users are editing a shared artifact, such as when a user performs an operation that adds or edits a translatable string. For example, a user edits a field's display label or help text, or a validation rule's error message, while another user performs an operation around the same time that similarly affects translatable strings. Another example of a shared artifact conflict is when two or more users are working in navigator menus which are shared across applications.

- **Conflicts between sandboxes intended for publishing:** Multiple sandboxes with the same customized artifact publishing to the mainline.

  Conflicts between sandboxes can arise when there is more than one sandbox intended for publishing in use. If two sandboxes contain conflicting customization changes to the same artifact and both are being published, then the sandbox that is being published last will not be allowed to be published, and an error describing the conflict will be displayed. To avoid such conflicts, it is recommended to create and use only one sandbox at a time. These types of conflicts can also occur with shared metadata files such as resource bundles that store translatable strings.

When multiple users are working in a single sandbox, these guidelines must be followed:

- Multiple concurrent users in the same sandbox must operate only on different and unrelated objects. For example, if user1 updates object1, then user2 can update object2 but should not update object1. Be aware that if both modifications involve changes to translatable strings, then saving changes to separate objects around the same time may still cause a conflict in the resource bundle that stores the translatable strings.
- Users in the same sandbox can see the changes created by one another. The latest version of each object gets loaded on-demand the first time it is viewed. If there are ADF Business Components customizations, then users must log out and log in again to see those changes reflected in the UI.

When multiple users are working in multiple sandboxes, in addition to all guidelines applicable to multiple users working in a single sandbox, these guidelines must be followed:

- There can be any number of test-only sandboxes operating concurrently. Multiple users can use multiple sandboxes concurrently for testing even if these sandboxes are never published. Sandboxes that are used for testing only, and that are not published, cause no conflicts with each other, but all guidelines for multiple users working in a single sandbox must be followed. However, all modifications are lost when the sandboxes are deleted.
- For sandboxes that will be published, you can have multiple concurrent sandboxes only if they operate on mutually exclusive artifacts. For example, you can have one sandbox that contains a page that is being customized to add a task flow, and another sandbox that contains a different page from a different application.
- If an artifact is updated in both the mainline and in the sandbox (or two different sandboxes), when the sandbox is published, such conflicts are detected and an error is generated.

19.1.1.1 Troubleshooting Concurrency Issues

Table 19–1 lists the issues that you might encounter if there are concurrency conflicts in the sandbox usage and the possible solutions.

Table 19–1 Troubleshooting Concurrency Issues

Example Scenario	Problem	Solution
Working on multiple sandboxes intended for publishing concurrently: Create sandbox S1, create sandbox S2, make changes to S2, publish S2, make changes to S1, and publish S1.	When you try to publish S1, an error is thrown.	Create a new sandbox and redo the changes.
Migrating sandboxes out-of-order: In environment 1, create sandbox S1, make changes to S1, export and publish S1. Repeat the same for S2. In environment 2, import S2, publish S2. Then, import S1,and publish S1. Sandboxes S1 and S2 are published in different order.	If there is any overlap between S1 and S2, for example both sandboxes updated the same MDS document), then changes made as part of S2 are overwritten by S1. For example, if AD connector form is created as part of S1 and EBS connector form is created as part of S2, then there will be overlap in CatalogAM.xml.xml and BizEditor resource bundle file. After the migration, both CatalogAM.xml.xml and BizEditor resource bundle only contain changes for AD Connector developed as part of S1.	Publish the sandboxes in correct order. You will be able to republish them.
Skipping sandbox during migration: In environment 1, create sandbox S1, make changes to S1, export and publish S1. Repeat the same for S2. In environment 2, import S2, publish S2. Do not migrate S1 at all. S1, which is published in environment 1, is not migrated to environment 2.	If S2 depends on changes made as part of S1, then those changes will be missing in environment 2.	Publish both sandboxes. You will be able to re-publish them.
Migrating sandboxes from multiple source environments: In environment 1, create sandbox S1, make changes to S1, export and publish S1. In environment 2, create sandbox S2, makes changes to S2, export and publish S2. In environment 3, import S1, publish S1. Import S2, and publish S2.	If there is any overlap between S1 and S2, for example both sandboxes updated the same MDS document, then changes made as part of S1 will be lost. For example, if AD connector form is created as part of S1 and EBS connector form is created as part of S2, then there will be overlap in CatalogAM.xml.xml and BizEditor resource bundle file. After the migration, both CatalogAM.xml.xml and BizEditor resource bundle only contain changes for EBS Connector developed as part of S2.	Manually merge the sandboxes into one.

19.1.2 Creating a Sandbox

To create a sandbox:

1. Log in to Oracle Identity Self Service or Oracle Identity System Administration.
2. On the upper navigation bar, click **Sandboxes**. The Manage Sandboxes page is displayed. This page has the following sections:
   - **Available Sandboxes:** Displays all the sandboxes that are available for testing the UI customizations, which are not yet published.

- **Published Sandboxes:** Displays all the published sandboxes.

3. On the toolbar, click **Create Sandbox**. The Create Sandbox dialog box is displayed.
4. In the Sandbox Name field, enter a name for the sandbox. This is a mandatory field.
5. In the Sandbox Description field, enter a description of the sandbox. This is an optional field.
6. Click **Save and Close**. A message is displayed with the sandbox name and creation label.

> **Caution:** Selecting the Activate Sandbox option closes all the open tabs except the Manage Sandboxes tab and activates the created sandbox.

7. Click **OK**. The sandbox is displayed in the Available Sandboxes section of the Manage Sandboxes page.

19.1.3 Activating and Deactivating a Sandbox

To activate a sandbox:

1. From the table showing the available sandboxes in the Manage Sandboxes page, select the sandbox that you want to activate.
2. On the toolbar, click **Activate Sandbox**.

   The table refreshes and a marker in the Active column is displayed. In addition, the Sandboxes link on the upper navigation bar also displays the active sandbox name in parentheses.

> **Caution:** If any other tabs are open except the Manage Sandboxes tab before activating the sandbox, then Oracle Identity Manager prompts that all the tabs will be closed before the sandbox can be activated.

To deactivate a sandbox:

1. From the table showing the available sandboxes in the Manage Sandboxes page, select the active sandbox that you want to deactivate.
2. On the toolbar, click **Deactivate Sandbox**. The page refreshes and the marker in the Active table disappears.

> **Caution:** If any other tabs are open except the Manage Sandboxes tab before deactivating the sandbox, then Oracle Identity Manager prompts that all the tabs will be closed before the sandbox can be deactivated.

19.1.4 Viewing and Modifying Sandbox Details

To view the details of a sandbox and modify the details:

1. In the table showing the available sandboxes in the Manage Sandboxes page, click the sandbox name link. A dialog box with the sandbox details is displayed.

2. Make the following changes:
   - In the Description field, you can enter a description for the sandbox.
   - View all the changes to the sandbox in the Change Details table.
   - Filter sandbox changes by using the Layer Names, Layer Values, and Change Types lists, and the Filter toolbar icon.
   - Delete any changes made in the sandbox by selecting the change in the table, and clicking **Delete Customization**.
   - Export the sandbox, if it contains any changes, by clicking **Export Sandbox**.

19.1.5 Exporting and Importing a Sandbox

To export a sandbox from an Oracle Identity Manager deployment to another:

1. From the table showing the available sandboxes in the Manage Sandboxes page, select the sandbox that you want to export.
2. On the toolbar, click **Export Sandbox**.

   If the sandbox contains any changes, then the sandbox content ZIP file starts downloading. You can now take the ZIP file and import it to the same or another environment.

Note: The name of the sandbox ZIP file is not the sandbox name. The sandbox name usually starts with IdM_ and it is specified in the XML file located inside the ZIP in the /mdssys/sandbox/ directory.

Caution: If the deployment on which the sandbox content ZIP file is being imported already contains a sandbox with the same name, then that sandbox will get overwritten.

To import a sandbox from an Oracle Identity Manager deployment to another:

1. On the toolbar, click **Import Sandbox**. The Import Sandbox dialog box is displayed.
2. In the Sandbox Archive field, enter a path to the sandbox archive that you exported.
3. Click **Import**.
4. Click Refresh. The sandbox, which is imported to the target deployment, is displayed in the Available Sandboxes tab.

19.1.6 Publishing a Sandbox

To publish a sandbox:

Note:

- Make sure that you export the sandbox before publishing it. After the sandbox is published, it cannot be exported anymore, and therefore, there is no way to migrate it to another environment.
- Oracle recommends creating a backup of MDS before publishing the sandbox. A backup of MDS can be created by using Oracle Enterprise Manager. See "Creating MDS Backup" on page 24-2 for information about creating a backup of the MDS by using Oracle Enterprise Manager.

---

1. From the table showing the available sandboxes in the Manage Sandboxes page, select the sandbox that you want to publish.
2. On the toolbar, click **Publish Sandbox**. A message is displayed asking for confirmation.
3. Click **Yes** to confirm. The sandbox is published and the customizations it contained are merged with the main line.
4. You can click the **Published Sandboxes** tab to view a list of the published sandboxes.

19.1.7 Publishing Sandboxes in Bulk and Sequence

To publish sandboxes in bulk and sequence:

1. Create a text file with the names of the sandboxes in the sequence in which you want to publish them, separated by commas. Save the file.
2. In the Manage Sandboxes page, click **Bulk Publish**. The Bulk Import and Publish dialog box is displayed.

Note: The **Bulk Publish** button is available in the Manage Sandboxes page only if you have applied Bundle Patch 11.1.2.3.170418.

---

3. In the Sandbox Archive section, click **Browse**, navigate to the directory that contains the sandbox zip file, and select the file to include it in the list of sandboxes to be published. Repeat the selection for all the sandboxes that you want to publish.
4. Click **Browse** adjacent to the Sandbox file to be uploaded field, and select the text file with the sandbox names and sequence that you saved in step 1.
5. Click **Generate Sequence**. The sandbox names along with the sequence in which they will be published as specified in the text file are displayed in the Sandbox Sequence field.
6. In the Sandbox Sequence field, review the sandbox names to be published and the sequence in which they will be published.
7. If you want to change the sandbox archives or the sequence, then edit the text file with the correct sandbox names and sequence, select it in the Sandbox file to be uploaded field, and click **Generate Sequence** again. Otherwise, click **Publish**.

19.1.8 Deleting a Sandbox

To delete a sandbox:

1. From the table showing the available sandboxes in the Manage Sandboxes page, select the sandbox that you want to delete.
2. On the toolbar, click **Delete Sandbox**. A message is displayed asking for confirmation.
3. Click **Yes** to confirm. The sandbox is deleted and is no longer displayed in the Manage Sandboxes page.

Note: Deleting a sandbox does not delete the forms created while the sandbox is active. Deleting forms is not supported in this release of Oracle Identity Manager.

19.1.9 Reverting Changes to Default Settings

You must perform all customizations within a sandbox. Until the sandbox is published, the changes are visible to you only and can be easily reverted by deactivating or deleting the sandbox. After the sandbox is published, the changes done cannot be reverted.

You can remove specific changes from the sandbox in any one of the following ways:

- Export the sandbox and modify it manually.
- Navigate to the Manage Sandboxes page, open the details of your sandbox, select a change, and delete it by clicking **Delete Customization**.

When an MDS sandbox is published, the documents are committed to the MAIN line. Your application starts using these documents immediately, and the application user views the effect of publishing the sandbox. Sometimes, you might inadvertently publish an incomplete or a wrong sandbox. In such instances, it is possible to recover your MAIN line to the state just before you created the wrong sandbox.

For example, if you create a sandbox called ShowAdminFeature at time T1, and in that you customized a JSFF fragment published at time T2. You realize later that the sandbox you published is wrong, and you want to recover your state to time T1. Also, if you are unable to login to Oracle Identity System Administration after customizing the interface and publishing the sandbox, then perform the following steps:

1. Login to Oracle Enterprise Manager.
2. In Application Deployments, select **oracle.iam.ui.console.self-service.ear**.
3. On the top-right of the page, select **Application Deployment**, and then select **MDS Configuration** from the list.
4. At the bottom of the screen, select **Runtime MBean Browser** under the Advanced Configuration section. The right side of the screen refreshes.
5. Click the **Operations** tab.
6. Scroll down and identify the listMetadataLabels MBean operation and invoke it. Select the MBean operation that does not take any parameters. Select the sandbox precreate that you want to restore, and copy it to the clipboard.

   For example, the value you copy can be similar to: Creation_IdM_test_09:25:00.
7. Click **Return** to go back to the Operation tab.

8. Find the promoteMetadataLabel MBean operation.
9. Invoke the promoteMetadataLabel MBean operation, and enter the value that you copied in step 6.
10. Restart Oracle Identity Manager.
11. Login to Oracle Identity System Administration.

> **Note:** You can also restore to the last successful sandbox that was published by restoring to the post label of that sandbox.

19.2 Skin Customization in Oracle Identity Manager

Oracle ADF uses skins along with styles to customize the appearance of an application. These concepts apply to all the Oracle Identity Manager interfaces, with the exception of the Transitional UI popups.

> **See Also:** Before customizing style sheets, see Customizing the Appearance Using Styles and Skins in the *Fusion Middleware Web User Interface Developer's Guide* in the following URL:
>
> `http://docs.oracle.com/cd/E15523_01/web.1111/b31973/af_skin.htm#ADFUI330`
>
> Following URL gives a list of all the CSS style selectors that can be used to customize the style sheets:
>
> `http://download.oracle.com/docs/cd/E15523_01/apirefs.1111/e15862/toc.htm`

19.2.1 Configuring a New Skin

You can extend `oim-alta` skin to configure custom skins. The `oim-alta` skin is shipped with Oracle Identity Manager. The custom skin files and skin definition are deployed as part of the `oracle.iam.ui.custom-dev-starter-pack.war` shared library.

To create, deploy and configure a custom skin that extends `oim-alta` skin:

1. Create META-INF directory with trinidad-skins.xml file, as shown:

```
<?xml version="1.0" encoding='utf-8'?>
<skins xmlns="http://myfaces.apache.org/trinidad/skin">
    <skin>
        <id>my-skin.desktop</id>
        <family>my-skin</family>
        <version>
            <name>v1</name>
            <default>true</default>
        </version>
        <extends>oim-alta.desktop</extends>
        <render-kit-id>org.apache.myfaces.trinidad.desktop</render-kit-id>
        <style-sheet-name>skins/my-skin/my-skin.css</style-sheet-name>
  </skin>
</skins>
```

2. Create `META-INF/skins/my-skin/my-skin.css`, and add your custom skin selectors, as described in section "Customizing the Appearance Using Styles and Skins" in *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

3. Create a JAR file that contains the `META-INF` directory. For example, you can use the following command to create the JAR file:

   ```
   jar cf skins.jar META-INF
   ```

   You must run the command from the parent directory of the `META-INF` directory. If the skin is referencing custom images or other files, then include them in the JAR file as well.

4. Include the newly created JAR file in the `WEB-INF/lib/` directory of the `oracle.iam.ui.custom-dev-starter-pack.war` shared library. You can find the deployed version of the `oracle.iam.ui.custom-dev-starter-pack.war` shared library in the *IDM_HOME*/server/apps/ directory. For example, you can use the following commands to update the existing `oracle.iam.ui.custom-dev-starter-pack.war` and include the additional JAR file:

   ```
   mkdir -p WEB-INF/lib
   cp skins.jar WEB-INF/lib
   jar uf oracle.iam.ui.custom-dev-starter-pack.war WEB-INF/lib/skins.jar
   ```

5. Copy the updated `oracle.iam.ui.custom-dev-starter-pack.war` to the *IDM_HOME*/server/apps/ directory.

6. Login to Oracle Identity System Administration, and change the values of the following system properties:

   Skin Family for OIM UI: Change the value to `my-skin`, or whatever value you specified for family in the trinidad-skins.xml file.

   Skin Version for OIM UI: Change the value to `v1`, or whatever value you specified for version in the trinidad-skins.xml file.

7. Restart Oracle Identity Manager server.

19.2.2 Changing Branding and Logo

Customizing or changing UI artifacts, such as logo, buttons, and menu items, can be done at runtime.

Note: The procedure documented in this section is for changing the branding and logo by customizing Oracle Identity Self Service. If you want to customize UI artifacts of the window that opens from the Oracle Identity System Administration (also referred to as the legacy Advanced Console), for example, the window that opens when you click **Configuration Properties** under System Configuration, then see:

`http://docs.oracle.com/cd/E21764_01/doc.1111/e14309/uicust.htm#BABFCFID`

To change the logo image:

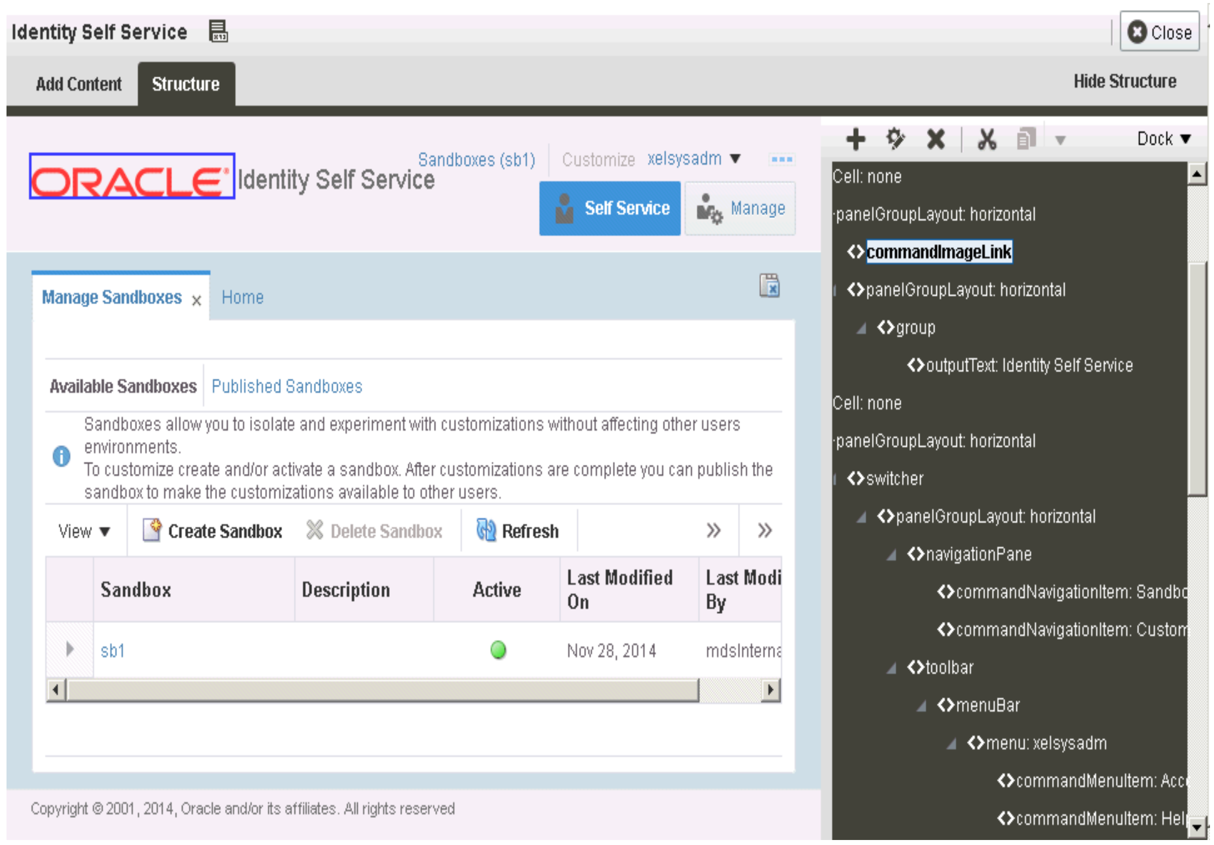
1. Log in to Oracle Identity Self Service as the system administrator.
2. Create and activate a sandbox.

Note: Creating and activating a sandbox is mandatory for customizing the UI by using the Web Composer. Without an active sandbox, Oracle Identity Manager does not allow to open any page in customization mode.

---

3. Click **Customize**. The customization panel is displayed at the top of the page.
4. Click **Structure**. The component tree is displayed. The component tree shows all the ADF components of the page.
5. Click the logo. The Confirm Shared Component Edit dialog box is displayed asking for confirmation.
6. Click **Edit**. The logo object is selected in the component tree, as shown in Figure 19–1:

Figure 19–1 The Object Library in WebCenter Composer



Sandboxes allow you to isolate and experiment with customizations without affecting other users environments.
To customize create and/or activate a sandbox. After customizations are complete you can publish the sandbox to make the customizations available to other users.

Sandbox	Description	Active	Last Modified On	Last Modi By
sb1		●	Nov 28, 2014	mdsInterna

Copyright © 2001, 2014, Oracle and/or its affiliates. All rights reserved

7. Click the icon. The Component Properties dialog box is displayed.
8. In Component Properties, click the down arrow icon next to the `Icon` property, and select **Expression Builder**.
9. In the Expression Builder, replace the default value of `#{attrs.logoImagePath}` with your logo path or URL.

Tip:

- Customizing the default EAR and WAR files, such as Self Service EAR, System Administration EAR, and xlWebApp.war, is not supported.
- By default, the Oracle logo is 119x25 pixels. Therefore, you can use a custom logo of the same dimensions. If you want a bigger logo, then it requires CSS changes.
- If you want to specify a font for any ADF component by using the Style tab of the Component Properties dialog box, then ensure that your target browsers and platforms support that specific font name. To look at the supported list for Mozilla Firefox, select **Tools**, **Options**, **Content**, **Fonts and Colors**. For Microsoft Internet Explorer, select **Tools**, **Internet Options**, **General**, **Fonts**.

10. Click **Apply**. The logo has changed to the new one you specified.
11. To change the Identity Self Service global banner, click the **Identity Self Service** text, and open the Component Properties dialog box.

    Tip: To change the banner in the Oracle Identity Manager login page, you must open the login page in the customization mode. However, the Customize link is not available in the login page. Therefore, to open the login page in customization mode:

    1. Login to Oracle Identity Self Service as an administrator with privileges to customize the UI.
    2. In an active sandbox, click the **Customize** link. The Oracle Identity Self Service is in customization mode.
    3. Perform the steps described in "Customizing Unauthenticated Pages" on page 19-22.

12. In the Display Options tab of the Component Properties dialog box, click the down arrow next to the Value field, and select **Expression Builder**. The Expression Editor dialog box is displayed.
13. With the **Type a value or expression** selected, enter a text to replace Identity Self Service, and click **OK**.
14. Click **Apply**.
15. Click **Close** to close WebCenter Composer.
16. Publish the sandbox.

19.3 Customizing Pages at Runtime

Customizing Oracle Identity Manager can be broadly categorized into customizing the UI and extending the object definitions of the user, role, organization, catalog, and provisioning target resource entities.

Table 19–2 lists the artifacts that can be customized for each entity.

Table 19–2 Entity Artifacts for Customization

Enity	Artifacts
User	Create Page
	Modify Page
	User Attribute Details
	Advanced Search Interface
	My Information
	Self Registration
Role	Create Page
	Modify Page
	Advanced Search Interface, which includes:
	- Query Criteria
	- Results Table columns
Organization	Create page
	Modify Page
	Advanced Search interfact, which includes:
	- Query Criteria
	- Results Table columns
Catalog	Catalog Search Page that includes:
	- Results Table columns
	- Catalog Item Details
Provisioning target resource	Provisioning Target Resource Create Form
	Provisioning Target Resource Modify Form
	Provisioning Target Resource Bulk Form

See Also: "Managing Forms" and "Configuring Custom Attributes" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager* for information about creating and managing forms by using the Form Designer

This section contains the following topics:

- Using Expression Language in UI Customization
- Showing or Hiding UI Components Conditionally
- Showing Request Profiles Conditionally
- Validating Input Data Using ADF Validators
- Marking Input Attribute as Required

The Web Composer enables you to customize the UI at runtime. This section describes the following UI customizations:

- Adding a Link or Button
- Hiding and Deleting an ADF Component
- Showing and Hiding Attributes
- Customizing Unauthenticated Pages

- Customizing Certification Pages

19.3.1 Using Expression Language in UI Customization

Expression Language (EL) allows you to access application data stored in JavaBeans components. For an introduction to EL and EL expression syntax, refer to the following URL:

`http://docs.oracle.com/javaee/6/tutorial/doc/gjddd.html`

19.3.1.1 Available EL Expressions in the User Context

The OIMContext bean is defined as an ADF session-scoped bean and provides access to information about the logged-in user. Table 19–3 lists the available EL expressions in the Oracle Identity Manager user context.

Table 19–3 EL Expressions in User Context

EL	Description
`#{oimcontext.currentUser['ATTRIBUTE_NAME']}`	Access value of the *ATTRIBUTE_NAME* attribute of the logged-in user.
`#{oimcontext.currentUser['UDF_NAME']}`	Access value of the *UDF_NAME* attribute of the logged-in user. UDF attributes can be defined by using the Form Designer.
`#{oimcontext.currentUser.roles}`	Access the *ROLE_NAME* and RoleEntity mapping that contains the roles assigned to the logged-in user. RoleEntity is Java Bean having name, description, key, and displayName properties.
`#{oimcontext.currentUser.roles['SYSTEM ADMINISTRATORS'] != null}`	Boolean EL that evaluates to true if the logged-in user has the System Administrator admin role. Similarly, you can modify the EL to check for any other role.
`#{oimcontext.currentUser.adminRoleMap['OrclOIMSystemAdministrator'] != null}`	Boolean EL that evaluates to true if the logged-in user has the OrclOIMSystemAdministrator admin role. Similarly, you can modify the EL to check for any other admin role.

You can use EL expression to retrieve all available user attribute values from the oimcontext bean, as shown in the following examples:

- To get the user key of the currently logged-in user:

  ```
  #{oimcontext.currentUser.usr_key}
  ```

  OR:

  ```
  #{oimcontext.currentUser['usr_key']}
  ```

- To get the list of role names of the currently logged-in user:

  ```
  #{oimcontext.currentUser.roles}
  ```

- To get the list of admin role names of the currently logged-in user:

  ```
  #{oimcontext.currentUser.adminRoles}
  ```

As an example, if you want to display a message with the user login name when a user logs in to Oracle Identity Self Service, then you can use EL expression to retrieve the login name of the currently logged-in user, and display it on the page. The expression to retrieve the user login name is the following:

```
#{oimcontext.currentUser['User Login']}
```

19.3.1.2 Available EL Expressions in the RequestFormContext

RequestFormContext is a bean available in the pageFlowScope of entity form details task flow. The entity forms include user form, application instance form, role form, and entitlement form. RequestFormContext provides various context information. Using this context information, you can customize the forms based on specific business requirements.

Table 19–4 lists the EL expressions involving RequestFormContext.

Table 19–4 EL Expressions in RequestFormContext

EL	Description
`#{pageFlowScope.requestFormContext}`	Access current instance of RequestFormContext.
`#{pageFlowScope.requestFormContext.operation}`	Access operation type that is being performed on the entity. The possible values are CREATE, MODIFY, ENABLE, DISABLE, and REMOVE.
`#{pageFlowScope.requestFormContext.operation == 'MODIFY'}`	Boolean EL that evaluates to true if current operation being performed on the entity is MODIFY.
`#{pageFlowScope.requestFormContext.actionType}`	Access action that is being performed by the user when the entity form is displayed. The possible values are APPROVAL, FULFILL, REQUEST, VIEW, and SUMMARY.
`#{pageFlowScope.requestFormContext.actionType == 'REQUEST'}`	Boolean EL that evaluates to true if the action that is being performed by the user when the entity form is displayed is REQUEST, for example, requesting role or application instance.
`#{pageFlowScope.requestFormContext.bulk}`	Boolean EL that evaluates to true if the operation being performed is a bulk operation, for example, requesting multiple application instances at a time.
`#{pageFlowScope.requestFormContext.beneficiaryIds}`	Access the list of beneficiary or target user IDs. For example, if you are requesting an application instance for user John Doe, then the list contains the user ID of John Doe. **Note:** Oracle recommends accessing the list and performing operations on it by using Java code.
`#{pageFlowScope.requestFormContext.cartItemIds}`	Access the list of cart item IDs. For example, if you are requesting an application instance for a user, then the list contains the application instance ID that is being requested. **Note:** Oracle recommended accessing the list and performing operations on it by using Java code.
`#{pageFlowScope.requestFormContext.requestEntityType}`	Get entity type being requested. The possible values are ROLE, ENTITLEMENT, APP_INSTANCE, and USER.
`#{pageFlowScope.requestFormContext.requestEntityType == 'APP_INSTANCE'}`	Boolean EL that evaluates to true if the entity type being requested is APP_INSTANCE.

Table 19–4 (Cont.) EL Expressions in RequestFormContext

EL	Description
`#{pageFlowScope.requestFormContext.requestEntitySubType}`	Access subtype of entity being requested. For example, when requesting APP_INSTANCE, requestEntitySubType is the application instance key.
`#{pageFlowScope.requestFormContext.instanceKey}`	Access the key of the instance being modified.

19.3.1.3 Internationalization for Resource Strings

In Oracle Identity Manager, you can create custom resource bundles and reference them in the UI. If you want to modify some of the predefined UI elements such as labels, headers, and so on, or the values displayed on a certain page (for example, values displayed in the Status field of the Request Summary page), then perform the procedure described in this section.

To create custom resource bundles:

1. Open the `oracle.iam.ui.custom-dev-starter-pack.war` shared library. The deployed version of the library is in the *IDM_HOME*/server/apps/ directory.
2. Create a new CustomResourceBundle.properties file.
3. In the new file, enter the key value pairs, for example:

   ```
   CUSTOMRB_BANNER_TEXT=My Identity and Access
   ```

4. Create all localized files, for example CustomResourceBundle_it.properties and CustomResourceBundle_es.properties, in the same directory.
5. Repackage the custom WAR, and update the custom WAR deployment in the server.

To use the resource bundles:

1. In Oracle Identity Self Service, create a sandbox, and click **Customize**.
2. On the Component Properties dialog box, open the Expression Editor for the specific property, and specify the expression, for example:

   ```
   #{adfBundle['oracle.iam.ui.custom.CustomResourceBundle'].CUSTOMRB_BANNER_TEXT}
   ```

3. Click **Test** to test the expression. Click **OK**, then click **Apply**.
4. Click **OK** to close the Component Properties dialog box.
5. Export the sandbox, and then publish the sandbox.

Note: Exporting the sandbox is optional, but it is a recommended step.

19.3.2 Showing or Hiding UI Components Conditionally

To conditionally show or hide UI components, use the rendered property of the component and assign EL expression to it that evaluates to Boolean. If the EL expression evaluates to true, then the component is shown. Consider the following examples:

Note: The rendered property of the component corresponds to the Show Component option in Oracle Web Composer.

- To show a UI component if the logged-in user has the System Administrators admin role:

  ```
  #{oimcontext.currentUser.roles['SYSTEM ADMINISTRATORS'] != null}
  ```

  Similarly, the EL expression can be modified to check if the logged-in user has any other role.

- To show a UI component if signed-in user has the System Administrator admin role:

  ```
  #{oimcontext.currentUser.adminRoles['OrclOIMSystemAdministrator'] != null}
  ```

  Similarly, the EL expression can be modified to check if the logged-in user has any other admin role.

- To show a UI component if the usr_key attribute of the logged-in user is 1:

  ```
  #{oimcontext.currentUser['usr_key'] == 1}
  ```

- To show a UI component if the logged-in user's last name is Doe:

  ```
  #{oimcontext.currentUser['Last Name'] == 'Doe'}
  ```

- To show a UI component if the logged-in user belongs to the Xellerate Users organization:

  ```
  #{oimcontext.currentUser['Organization Name'] == 'Xellerate Users'}
  ```

- To show a UI component if the user's UDF attribute called UDF_NAME equals to UDF_VALUE:

  ```
  #{oimcontext.currentUser['UDF_NAME'] == 'UDF_VALUE'}
  ```

Note: "Showing Components Conditionally" on page 19-38 describes showing components based on certain conditions by implementing custom Managed Bean.

19.3.3 Showing Request Profiles Conditionally

To show a catalog request profile conditionally:

1. Login to Oracle Identity Self Service.
2. Activate a sandbox.
3. In the Self Service tab, click the **Request Access** box, and select **Request for Self**. The Add Access page of the Request Access wizard is displayed.
4. Click **Customize**. Click **Structure**. The component tree is displayed.
5. Using the component tree, navigate to the iterator component within Request Profiles. The iterator component has panelGroupLayout subcomponent, which represents single request profile.

6. Select **panelGroupLayout:horizontal**, which is a subcomponent of panelGroupLayout:vertical inside the iterator component, and click **Edit** in the Web Composer.
7. Assign a Boolean EL expression to the rendered property. This is the Show Component in Web Composer.

   For example, if you want to display a resource profile called Profile to users of the Suppliers organization only, and display any other profile to other users, then use the following expression:

   ```
   #{(row.profileName == 'Profile' && oimcontext.currentUser['Organization Name']
   == 'Suppliers') || row.profileName != 'Profile'}
   ```

   The EL expression is evaluated for every profile which is available. Similarly, you can modify/extend the EL expression to conditionally display any other profile.
8. Publish the sandbox to globalize the change.

19.3.4 Validating Input Data Using ADF Validators

To validate input component data using predefined ADF validators, you must modify the JSFF page fragment and include one of the ADF validators as a child element of input component. Table 19–5 lists the ADF validators:

Table 19–5 ADF Validators

Validator	Description
<af:validateByteLength>	Validates the byte length of strings when encoded
<af:validateDateRestriction>	Validates that the date entered is within a given restriction
<af:validateDateTimeRange>	Validates that the date entered is within a given range
<af:validateDoubleRange>	Validates that the date entered is within a given range
<af:validateLength>	Validates that the value entered is within a given length
<af:validateLongRange>	Validates that the value entered is within a given range
<af:validateRegExp>	Validates an expression by using Java regular expression syntax

For example, to validate that the only allowed characters for the User Login attribute are alphanumeric ASCII characters, you can include the following RegExp validator as a child element of the User Login input component:

```
<af:validateRegExp pattern="[a-zA-Z0-9]*"/>
```

ADF validators cannot be added directly by using the Web Composer. Instead, you can add another component as a child component of the User Login component, for example, another input text. After that you can export the sandbox containing this change. Finally, update the JSFF page fragment for the form in the exported sandbox, and then import the sandbox.

Note: "Implementing Custom Field Validation" on page 19-43 describes implementing the custom field validator by using custom Managed Bean.

19.3.5 Marking Input Attribute as Required

To conditionally make an input field required, you can use the required property of the component, and assign it a Boolean EL expression. If the EL expression evaluates to true, then the component is marked as required, and the required validation is triggered.

For example EL expressions, see "Showing or Hiding UI Components Conditionally" on page 19-17.

For more information about making field conditionally mandatory based on the value of another field, see "Setting a Conditional Mandatory Field" on page 19-41.

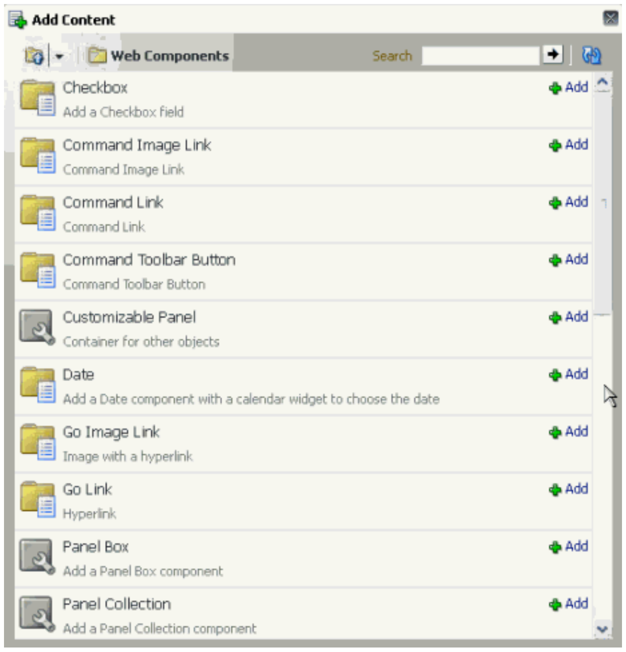
19.3.6 Adding a Link or Button

To add a link to Oracle Identity Self Service:

1. From any page in Oracle Identity Self Service, open WebCenter Composer.
2. Select the top panel on which you want to include the link. The ADF component is selected in the component tree.
3. Click the plus (+) icon to open the Add Content dialog box. Navigate and select the Web Components component from the list of components.

   The Web Components component in the Add Content dialog box is shown in Figure 19–2:

Figure 19–2 The Add Content Dialog Box



Checkbox	Add a Checkbox field	Add
Command Image Link	Command Image Link	Add
Command Link	Command Link	Add
Command Toolbar Button	Command Toolbar Button	Add
Customizable Panel	Container for other objects	Add
Date	Add a Date component with a calendar widget to choose the date	Add
Go Image Link	Image with a hyperlink	Add
Go Link	Hyperlink	Add
Panel Box	Add a Panel Box component	Add
Panel Collection	Add a Panel Collection component	Add

4. Search for the link component that you want to add, and click **Add** in the same row. The link is added to the selected panel.

Note: For a complete list of UI components, see *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

5. Click **Close** to quit customization mode.

Note: For more details, see the following sections:

- "Launching Taskflows" on page 19-49
- "Creating an External Link" on page 19-51

19.3.7 Hiding and Deleting an ADF Component

Hiding an ADF component results in the UI artifact being hidden from the user. To hide an ADF component:

1. In Oracle Identity Self Service, go to the page on which you want to hide a component.
2. Click **Customize** to open WebCenter Composer.
3. Click **Structure** to open the component tree.
4. Click the component on the page that you want to hide. The corresponding ADF component in the component tree is selected.
5. Right-click the selected ADF component in the component tree, and select **Hide**.

To delete an ADF component:

1. From the Oracle Identity Self Service page on which you want to delete any UI component, open Web Composer.
2. Click **Structure** to open the component tree.
3. Click the component on the page that you want to delete. The corresponding ADF component in the component tree is selected.
4. Right-click the selected ADF component in the component tree, and select **Delete**.
5. In the Delete Component Confirmation box, click **Delete**.

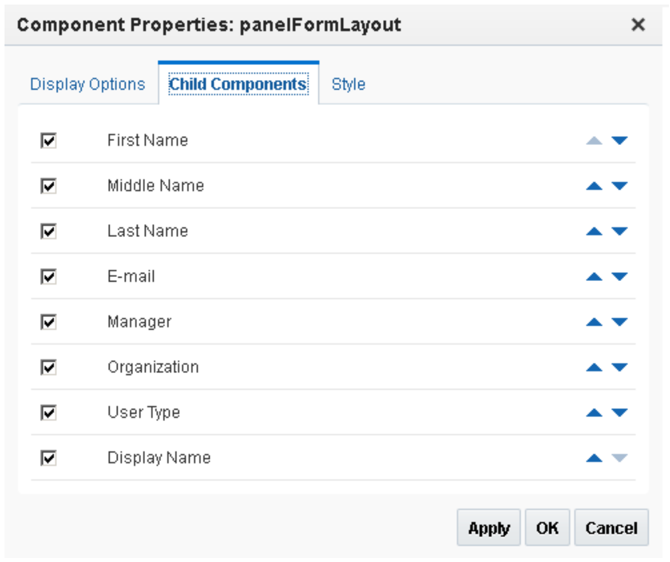
19.3.8 Showing and Hiding Attributes

To show or hide attributes in a page:

1. Go to the page on which you want to show or hide the attribute. For example, navigate to the My Information page in the Oracle Identity Self Service if you want to show or hide the Telephone field.
2. Click **Customize** to open Web Composer.
3. Click **Structure** to open the component tree.
4. Click the region or section that contains the attribute you want to hide, or you want the attribute to be shown.

   The Confirm Task Flow Edit message box is displayed.
5. Click **Edit**. The ADF component for the selected region is selected in the component tree.
6. Open the Component Properties dialog box.
7. Click the **Child Components** tab. All the UI components of the selected region are displayed. Figure 19–3 shows a sample Child Components tab in the Component Properties dialog box.

Figure 19–3 The Child Components Tab


Component Properties: panelFormLayout

Display Options | Child Components | Style

- ☑ First Name
- ☑ Middle Name
- ☑ Last Name
- ☑ E-mail
- ☑ Manager
- ☑ Organization
- ☑ User Type
- ☑ Display Name

8. Select or deselect the checkbox corresponding to the attributes to show or hide the attributes respectively.

Note: If you do not see an attribute listed here, then you must add the attribute into the form. See "Adding a Custom Attribute" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager* for details.

---

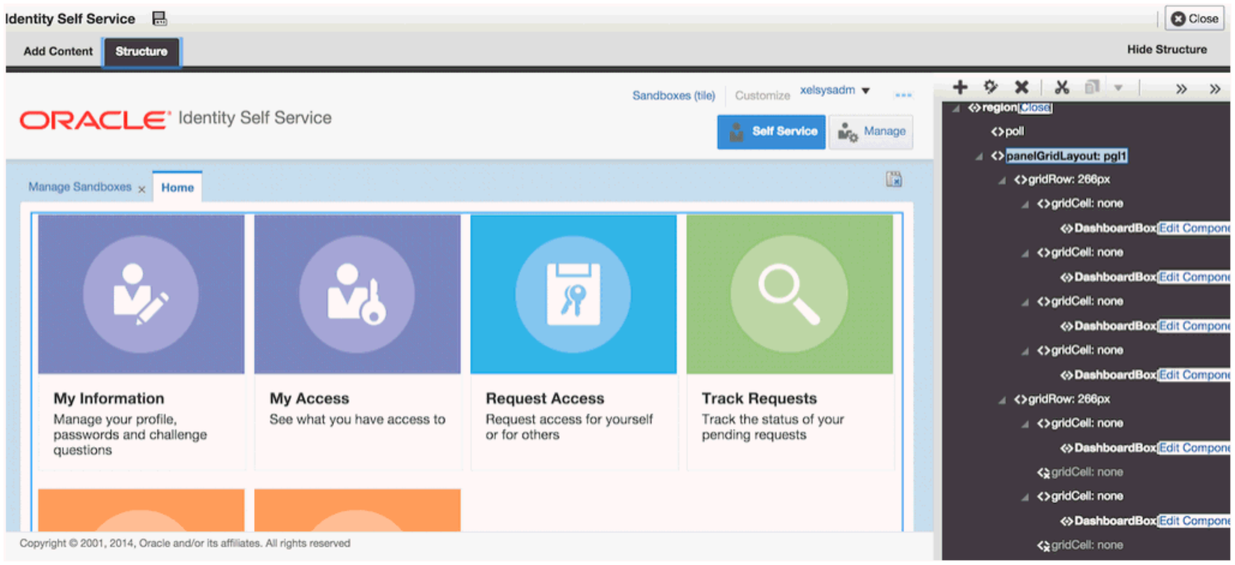
9. Click **Apply**. The selected attributes are hidden or shown based on your selection.
10. Click **OK**, and then click **Save** on the toolbar.

19.3.9 Customizing Unauthenticated Pages

To customize the unauthenticated pages of the Identity Self Service, such as User Registration page or Sign In page:

1. Login to Oracle Identity Self Service as the system administrator.
2. Create and activate a sandbox, and click **Customize**.
3. Click the **Self Service** tab to open the Self Service Home page.
4. Click Structure to open the component tree.
5. Select the **panelGridLayout** component, as shown in Figure 19–4.

Figure 19–4 The panelGridLayout Component


Identity Self Service
Close
Add Content
Structure
Hide Structure
Sandboxes (tile)
Customize
xelsysadm
ORACLE Identity Self Service
Self Service
Manage
Manage Sandboxes
Home
My Information
Manage your profile, passwords and challenge questions
My Access
See what you have access to
Request Access
Request access for yourself or for others
Track Requests
Track the status of your pending requests
Copyright © 2001, 2014, Oracle and/or its affiliates. All rights reserved
region Close
poll
panelGridLayout: pgl1
gridRow: 266px
gridCell: none
DashboardBox Edit Component

6. Select the last gridRow component. The component is shown as grayed out.
7. Right-click the last gridRow component, and select **Edit** to edit component properties.
8. Select the **Show Component** option, and click **Apply**.
9. Right-click the gridCell component under gridRow, and select **Edit**.
10. Select the **Show Component** option, and click **Apply**.
11. Right-click the group component under gridCell, and select **Edit**.
12. Select the **Show Component** option, and click **Apply**. A new home page tile named Unauthenticated Pages is displayed.
13. Click **Add Content** to switch to design view.
14. Click the Unauthenticated Pages tile. A menu is displayed. Each menu item represents a link to one of the unauthenticated pages. For example, click **New User Registration**, and you will be redirected to User Registration page.
15. After entering values in the required fields, click **Structure**. Select the **panelFormLayout** component, click **Add**, and use Data Component - User Registration, UserVO1 to add new fields.
16. Click the **Add Content** tab, and click **Cancel** to come back to the Home page.
17. After you are done, remember to hide the Unauthenticated Pages tile.

19.3.10 Customizing Certification Pages

This section describes how to customize the certification pages of the Oracle Identity Self Service. It contains the following sections:

- Customizing the Certification Detail Pane
- Adding Custom Attributes to the Certification Table
- Customizing the Certification Table

19.3.10.1 Customizing the Certification Detail Pane

The information from the row selected in the certification table can be used for customizing the detail pane found below the table. The procedure in this section can be used to customize the user certification detail pane. The same procedure can be followed for any certfication type.

After you have entered the customization mode, perform the following steps:

1. Edit the panelFormLayout containing the User Detail Information.
2. Click **Add Content**.
3. Select **Data Component - Certification**.
4. Select **UserCertificationUserVO1**.
5. Search for the attribute you want to add, for example Title, and click **Add**.
6. Select ADF Readonly Input Text with Label component.

   The input component is added to the page, but a value for it is not displayed. A label is added that shows the name of the attribute.
7. Select the inputText component in the page source panel, and click **Edit**. The Component Properties dialog box is displayed.
8. Scroll down and find the Value attribute, and open the Expression Builder.
9. Edit the expression value and set it to the following:

   ```
   #{pageFlowScope.p1_row_idcTitle}
   ```
10. Save the changes and close Web Composer. Select a row in the table.

    When a row is selected from the table, the information is stored in the pageFlowScope. To display this information in the detail pane, steps 1 through 10 must be followed to extract the correct data. The format of the EL to follow is:

    #{pageFlowScope.p1_row_*ATTRIBUTE_NAME*}

    Page 1 table information can also be used in page 2 by using the same format. Because the data is stored in the pageFlowScope, the information remains in the scope making it available for display. Page 2 has a Page 1 Detail section at the top of the page showing a reference back to the item on page 1. You can add more page 1 details here using p1_row_*ATTRIBUTE_NAME* in the expression.

    The steps documented in this section apply to page1 or the summmary page, of the current certification. If you want to customize page 2 or the detail page, then use the following format:

    #{pageFlowScope.p2_row_*ATTRIBUTE_NAME*}

19.3.10.2 Adding Custom Attributes to the Certification Table

To create a UDF and add it to the certification table:

1. Create and activate a sandbox.
2. Create a new user UDF, as described in "Creating a Custom Attribute" in *Administering Oracle Identity Manager*.
3. Publish the sandbox.
4. To add the UDF, create and activate a new sandbox.

5. Navigate to the Certification Dashboard, and open the certification detail page for an entity.
6. Click **Customize**. Click **Structure** to open the component tree.
7. Click the table on the certification detail page. Click Edit on the Confirm Task Flow Edit message box.
8. With the table selected on the component tree, click the plus (+) icon to open the Add Content dialog box.
9. Click Data Component - Certification.
10. Select the VO corresponding to the table, for example, **ApplicationCertificationEntitlementVO1**.

    The following table lists the VOs that are to be selected for various types of certifications:

Certification Page	VO
User Certification Phase 1 Page 1	UserCertificationUserVO
User Certification Phase 2 Page 2	UserCertificationPhase2EntitlementVO
Role Certification Detail Page	RoleCertificationMemberVO
Application Instance Certification Detail Page	ApplicationCertificationEntitlementVO
Entitlement Certification Detail Page	EntitlementCertificationEntitlementMemberVO

11. Scroll down to the UDF you created, and click **Add**. Then, select **ADF Table Column**.
12. Click **Close**. The UDF column is added to the certification table.

Note: To add a default attribute to the certification table, from the View menu, select **Columns**, ***ATTRIBUTE_NAME***. The default attribute column is added to the table. Similarly, you can hide the attribute from the certification table by selecting it from the **View**, **Columns**.

19.3.10.3 Customizing the Certification Table

To customize the certification table, for example, to increase the size of the table via customization:

1. Create and activate a sandbox.
2. Go to the certification detail page, and click **Customize**.
3. Click Structure to open the component tree.
4. Click anywhere on the certification table so that `table:t1` tag is selected on the component tree. Right-click **table:t1**, and select **Edit**. Alternatively, you can click the show properties icon on the toolbar of the component tree.

    The Component Properties dialog box is displayed.
5. Scroll down to the Fetch Size property. Click the down arrow of the Fetch Size property, and select **Expression Builder**.

6. In the Expression Builder, select the **Type a value or expression** option, and enter the value as `75`. Click **OK**.
7. Click **Apply**, and then click **OK** to close the Component Properties dialog box.
8. Click **Close**. The table has been expanded to 75 rows.

19.4 Securing a Task Flow Region Using EL Expressions

For each new task flow, there is an entry in the jazn-data.xml file, as shown in the following example:

```
<permission>
<class>oracle.adf.controller.security.TaskFlowPermission</class>
<name>/WEB-INF/oracle/iam/ui/catalog/tfs/request-summary-details-tf.xml#request-su
mmary-details-tf</name>
<actions>view</actions>
</permission>
```

This is the basic level of permission required for any task flow to be visible on the Identity Self Service UI. For advanced permissions dependent on admin roles, you can use EL expressions to enforce functional security.

For securing task flows, the task flow must be used as a region in the parent JSFF file. You can define EL expression for the region so that the task flow can be shown or hidden to the logged-in user based on the user's permissions.

For securing a region, consider the following example:

On the my-access-accounts.jsff page, the details-information-tf task flow is rendered selectively to the users by using the following EL expression:

```
rendered="# {oimappinstanceAuth.view [bindings.appInstanceKey].allowed}"
```

Here:

- oimappinstanceAuth is the mapped name of the ApplicationInstanceAuthz.java authorization bean in the adfc-config.xml file.
- view is the name of the UIPermission that needs to be checked. The following permission is defined in ApplicationInstanceAuthz.java, which is the actual bean file for reference of oimappinstanceAuth:

  ```
  Private UIPermission view = new UIPermission
  (PolicyConstants.Resources.APPLICATION_INSTANCE.getId(),
  PolicyConstants.ApplicationInstanceActions.VIEW_SEARCH.getId());
  ```

- appInstanceKey is the ID of the application instance that the user is trying to view, which is passed as a parameter.

19.5 Customizing Oracle Identity Manager Help

Oracle Identity Manager lets you develop and use the following online Help systems in the Oracle Identity Self Service and Oracle Identity System Administration:

- Adding Custom Help Topics
- Adding Inline Help

19.5.1 Adding Custom Help Topics

In addition to the Oracle Identity Manager help topics, you can also create and use custom help topics.

To view the custom help topics:

1. Login to Oracle Identity Self Service.
2. On the navigation bar at the top, click the down-arrow with the logged-in user name, and click **Help**. The Oracle Help for the Web window is displayed.
3. From the Book list, select **Custom Help Topics for Oracle Identity Manager**.
4. Expand the contents to view the help topics.

The custom help book is provided as a separate JAR file. This is the *OIM_HOME*/help/CUSTOMOHW.jar file. You can create your own help topics and custom help book JAR, and then replace the CUSTOMOHW.jar file to display your custom help topics in the UI.

You create the custom help topics by using Oracle Help for the Web (OHW). For detailed information about creating custom OHW help topics, see *Oracle Fusion Middleware Developer's Guide for Oracle Help*.

After creating the new custom help books, modify the following configuration files in the *OIM_HOME*/help/ directory to reference the new help books:

- **ohwconfig_identity.xml:** Configuration file for custom help topics in Oracle Identity Self Service
- **ohwconfig_sysadmin.xml:** Configuration file for custom help topics in Oracle Identity System Administration

Note: The configuration files are overwritten when you upgrade Oracle Identity Manager, and you must modify the configuration files again to reference the custom help books.

After creating the custom help topics, create the custom help JAR file, and replace the CUSTOMOHW.jar file with the new JAR file. You can now add your custom help topics on the UI pages. The following procedure shows how to add a custom help topic to the Home page in the Oracle Identity Self Service:

1. In Oracle Identity Self Service, activate a sandbox from the Manage Sandboxes page.
2. Go to one of the Home pages for Self Service, Compliance, or Manage, and click **Customize**.
3. Click **Structure** to open the component tree.
4. Click the section of the Home page where you want to add the help topic. Click **Edit** in the Confirm Edit Task Flow popup.
5. Click the plus (+) icon to open the Add Content dialog box.
6. Scroll down and click **Web Components**.
7. In the row for Command Image Link, click **Add**. The selected component is added to the Home page.
8. Select the added component, and click **Edit**. Open the Component Properties dialog box.

9. Click the **Display Options** tab.
10. In the Text field, enter the text for the help topic that will be displayed in the page.
11. In the Image field, enter the path and file name for the help icon image.
12. In the Action Listener field, enter the URL with the HelpTopicID of the custom help topic.
13. Click **Apply**, and then click **OK**.
14. Save and close customization mode. The help topic is added to the Home page. Clicking the help topic displays the help topic in the custom help book JAR file.

19.5.2 Adding Inline Help

Oracle Identity Manager does not provide inline help by default. However, you can add your inline help for the various UI components, such as add tooltip text for fields and buttons.

The content for the inline help is picked up from the files in the custom WAR library (oracle.iam.ui.custom-dev-starter-pack.war), such as the /oracle/iam/ui/custom/help/CustomHelpResourceBundle.properties file. If the CustomHelpResourceBundle.properties file is not available in the WAR library, then you can create it.

You can specify the inline help content through the entries in the CustomHelpResourceBundle.properties file. The entries have a CUSTOMRB prefix, and have any one of the following suffixes:

- _DEFINITION: This specifies inline help for a field or UI component. For example:

  CUSTOMRB_EMAIL_DEFINITION=Enter your official e-mail ID if available.

  EMAIL is the field name, and the value of the entry is the inline help text displayed on placing your mouse pointer on the field.

- _INSTRUCTIONS: This specified inline help for a page layout. For example:

  CUSTOMRB_MY_INFO_INSTRUCTIONS=Profile update will get reflected post approvals.

  MY_INFO is the page, and the value of the entry is the inline help text displayed on the top of the page.

As an example, the following procedure shows how to add inline help to the Telephone field in the My Information page of Oracle Identity Self Service:

1. In the Oracle Identity Self Service, navigate to the My Information page, and expand the Basic User Information section.
2. Click **Customize**, and open the component tree.
3. Click the Telephone field.
4. Click **Edit**, and open the Component Properties dialog box.
5. In the Help Topic ID field, enter the help topic ID of the inline help that you want to associate with the Telephone field, such as CUSTOMRB_TELEPHONE.

   Note that specifying the _DEFINITION suffix is not required.
6. Click **Apply**, and then click **OK**.

7. Save and close customization mode. An information image with the interrogation sign (?) is displayed before the Telephone field. When you place the mouse pointer on the icon, the inline help text is displayed.

 See Also: "Displaying Tips, Messages, and Help" on the *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework* for information about defining tips and messages and providing help information for ADF components

19.6 Customizing the Home Page

The Home pages provide a snapshot of the various functions in the Oracle Identity Self Service. You can customize the Home page by adding, removing, and rearranging containers or tiles.

This section describes how to customize the Home page. It contains the following topics:

- Adding a Tile to the Home Page
- Launching a New Page From the Tile Icon
- Launching a New Page From the Tile Menu
- Showing Tiles Conditionally

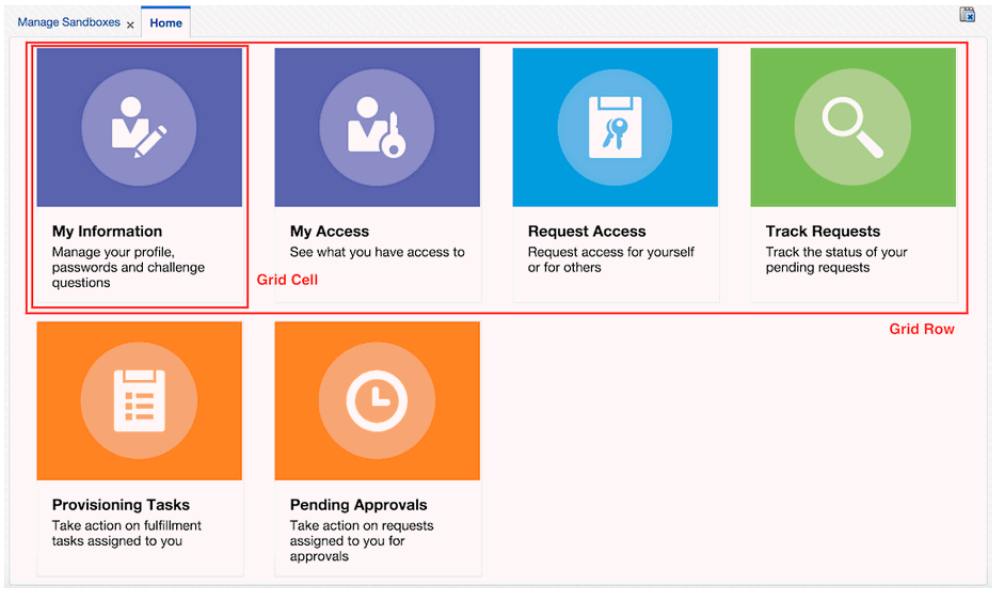
19.6.1 Adding a Tile to the Home Page

To add a tile to the Home page:

1. Create and activate a sandbox.
2. Navigate to the Home page to which you want to add a tile.

   Home pages consist of Panel Grid Layout, in which the page is divided into one or more grid rows and each grid row can have up to four grid cells. Figure 19–5 shows how the Home page is divided into grid rows and grid cells.

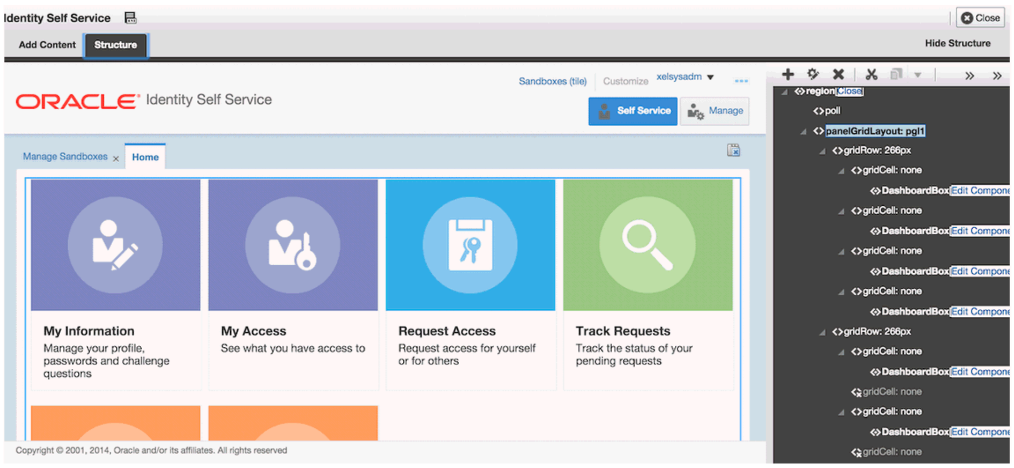
Figure 19–5 Home Page Panel Grid Layout


Manage Sandboxes
Home
My Information
Manage your profile, passwords and challenge questions
Grid Cell
My Access
See what you have access to
Request Access
Request access for yourself or for others
Track Requests
Track the status of your pending requests
Grid Row
Provisioning Tasks
Take action on fulfillment tasks assigned to you
Pending Approvals
Take action on requests assigned to you for approvals

3. Click **Customize** to switch to the customization mode.
4. Click **Structure** to switch to structure view.
5. Identify a position on the Home page where you want to add the new tile.

   Before you can add a new tile, a new Grid Cell and optionally a new Grid Row must be added.
6. To add a grid row:
   a. Select the Panel Grid Layout component as shown in Figure 19–6.

Figure 19–6 Panel Grid Layout Component


Identity Self Service
Close
Add Content
Structure
Hide Structure
ORACLE Identity Self Service
Sandboxes (tile)
Customize
xelsysadm
Self Service
Manage
Manage Sandboxes
Home
My Information
Manage your profile, passwords and challenge questions
My Access
See what you have access to
Request Access
Request access for yourself or for others
Track Requests
Track the status of your pending requests
Copyright © 2001, 2014, Oracle and/or its affiliates. All rights reserved
region Close
poll
panelGridLayout: pgl1
gridRow: 266px
gridCell: none
DashboardBox Edit Compone
gridCell: none
DashboardBox Edit Compone
gridCell: none
DashboardBox Edit Compone
gridCell: none
DashboardBox Edit Compone
gridRow: 266px
gridCell: none
DashboardBox Edit Compone
gridCell: none
gridCell: none
DashboardBox Edit Compone
gridCell: none

   b. Click the Add icon.
   c. In the resource catalog, go to Web Components, and click the **Add** link next to grid row component. A new grid row is added to the page as a first child of the panelGridLayout component.
   d. After adding the grid row component, you can change the values of Height, Margin Top, and Margin Bottom properties to align it with the existing grid rows.
7. To add a grid cell:
   a. Select the Panel Grid Layout component, as shown in Figure 19–6.
   b. Using the Component tree on the right, select one of the child grid rows where you want to add a new grid cell.
   c. Click the Add icon.
   d. In the resource catalog, go to Web Components, and click the **Add** link next to the grid cell component. A new grid cell is added to the page as a first child of the selected grid row component.
   e. After adding the grid cell component, you can change the values of the Align, Margin End, Margin Start, Width properties to align it with the existing grid cells.
8. After a new grid cell has been added, select it, and click the Add icon. In the resource catalog, go to Web Components, and click the **Add** link next to the Dashboard Box component. A new dashboard tile is added to the page.

Note: If the new tile is not displayed in the page, then refresh the page.

---

9. You can select the tile (DashboardBox component must selected in the Component tree), click the Edit icon, and change the values of the Hover Image, Image, Instruction Text, and Title Text properties.

After you add the tile, perform the steps described in one of the following sections to launch a new page by clicking on the tile. Do not publish your sandbox yet.

19.6.2 Launching a New Page From the Tile Icon

To launch a page by clicking on the tile icon:

1. Export the sandbox and unzip it.
2. Open the Home page jsff.xml file that you are working on by using a text editor. Oracle Identity Manager has the following default Home pages for Self Service, Manage, and Compliance:

```
oracle/iam/ui/homepage/home/pages/mdssys/cust/site/site/self-service-home.jsff.
xml
oracle/iam/ui/homepage/home/pages/mdssys/cust/site/site/self-service-manage.jsf
f.xml
oracle/iam/ui/homepage/home/pages/mdssys/cust/site/site/self-service-compliance
.jsff.xml
```

3. Locate the oim:DashboardBox element in the XML file. The element looks similar to the following:

```
<oim:DashboardBox xmlns:oim="/componentLib1" instructionText="My user details"
titleText="My Details" image="/images/Dashboard/myAccess.png"
hoverImage="/images/Dashboard/myAccess_s2.png" iconClickable="true"
id="e8533237995"/>
```

4. Ensure that the value of `iconClickable` is set to `true`.
5. Add a new element attribute named `iconClickAction`, set the value of the attribute to:

```
#{backingBeanScope.dashboardNavigationBean.launchTaskFlow}
```

6. Add two new `af:clientAttribute` elements as child elements of `oim:DashboardBox`, as follows:

```
<oim:DashboardBox xmlns:oim="/componentLib1" instructionText="My user details"
titleText="My Details" image="/images/Dashboard/myAccess.png"
hoverImage="/images/Dashboard/myAccess_s2.png" iconClickable="true"
id="e8533237995"
iconClickAction="#{backingBeanScope.dashboardNavigationBean.launchTaskFlow}">
<af:clientAttribute xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
name="taskFlowId"
value="/WEB-INF/oracle/iam/ui/manageusers/tfs/user-details-tf.xml#user-details-
tf"/>
<af:clientAttribute xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
name="title" value="My Details"/>

<!-- the following clientAttributes are optional, these are to pass values to
input parameters of user-details task flow -->
<af:clientAttribute xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
```

```
name="userLogin" value="#{oimcontext.currentUser['User Login']}"/>
<af:clientAttribute xmlns:af="http://xmlns.oracle.com/adf/faces/rich" name="
usr_key" value="#{oimcontext.currentUser['usr_key']}"/>
</oim:DashboardBox>
```

Make sure that `oim:DashboardBox` now has opening and closing tags, as shown in the example. Also, ensure that the component IDs are unique.

Set values of taskFlowId and title client attributes. taskFlowId specifies which task flow will be launched, and title specifies the title of the new tab.

You can add additional client attributes if you want to pass input parameters to the task flow, as shown in the example in this step.

> **Tip:** If all the required taskflow parameters are not available through EL expressions, then you can implement a custom actionListener, as described in "Launching Taskflows" on page 19-49. The new actionListener method will be accessible through an EL that must be set to iconClickAction property.
>
> If you want to launch some of the following UIs that are obsolete in this release of Oracle Identity Manager, then use the following ELs to set the iconClickAction property:
>
> - Attestation Dashboard:
>   `#{backingBeanScope.dashboardNavigationBean.navigateAttest ationDashboard}`
> - Pending Attestations:
>   `#{backingBeanScope.dashboardNavigationBean.navigatePendin gAttestations}`
> - Legacy Homepage:
>   `#{backingBeanScope.dashboardNavigationBean.navigateHome}`

7. Save the jsff.xml file, and re-create the sandbox ZIP file with the same name and structure as the original ZIP file.
8. Import the sandbox to Oracle Identity Manager.
9. Verify the changes and functionality of the new Home page tile.
10. Export the sandbox and publish it to make the changes available to all users.

19.6.3 Launching a New Page From the Tile Menu

To launch a page by clicking on tile menu item:

1. Export and unzip the sandbox.
2. Open the Home page jsff.xml file that you are working on by using a text editor. Oracle Identity Manager has the following default Home pages for Self Service, Manage, and Compliance:

```
oracle/iam/ui/homepage/home/pages/mdssys/cust/site/site/self-service-home.jsff.
xml
oracle/iam/ui/homepage/home/pages/mdssys/cust/site/site/self-service-manage.jsf
f.xml
oracle/iam/ui/homepage/home/pages/mdssys/cust/site/site/self-service-compliance
.jsff.xml
```

3. Locate the `oim:DashboardBox` element in the XML file. The element looks similar to the following:

```
<oim:DashboardBox xmlns:oim="/componentLib1" instructionText="Attestations"
titleText="Attestations" image="/images/Dashboard/myAccess.png"
hoverImage="/images/Dashboard/myAccess_s2.png" iconClickable="true"
id="e85332379959"/>
```

4. Ensure that `iconClickable` is set to `false`.
5. Add new popupMenu f:facet element as child element of `oim:DashboardBox`, as follows:

```
<oim:DashboardBox xmlns:oim="/componentLib1" instructionText="Attestations"
titleText="Attestations" image="/images/Dashboard/myAccess.png"
hoverImage="/images/Dashboard/myAccess_s2.png" iconClickable="false"
id="e85332379959">
<f:facet xmlns:f="http://java.sun.com/jsf/core" name="popupMenu">
<af:menu xmlns:af="http://xmlns.oracle.com/adf/faces/rich" id="m8278911">
<af:commandMenuItem xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
text="Attestation Dashboard"
actionListener="#{backingBeanScope.dashboardNavigationBean.navigateAttestationD
ashboard}"
id="cmi2478915"/>
<af:commandMenuItem xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
text="Pending Attestations"
actionListener="#{backingBeanScope.dashboardNavigationBean.navigatePendingAttes
tations}"
id="cmi2478916"/>
</af:menu>
</f:facet>
</oim:DashboardBox>
```

The menu can have as many command items as you want. The command menu items can use one of the following:

- Generic actionListener (#{backingBeanScope.dashboardNavigationBean.launchTaskFlow}) in conjunction with clientAttributes, as shown in "Launching a New Page From the Tile Icon" on page 19-31.
- Custom actionListener, as described in "Launching Taskflows" on page 19-49.
- One of the following actionListeners to launch interfaces that are obsolete in this release of Oracle Identity Manger, such as:
  - Attestation Dashboard: `#{backingBeanScope.dashboardNavigationBean.navigateAttestationDashboard}`
  - Pending Attestations: `#{backingBeanScope.dashboardNavigationBean.navigatePendingAttestations}`
  - Legacy Homepage: `#{backingBeanScope.dashboardNavigationBean.navigateHome}`

Make sure the `oim:DashboardBox` now has opening and closing tags, as shown in the example in this step. Also, ensure that component IDs are unique.

6. Save the jsff.xml, and re-create the sandbox ZIP file with the same name and structure as the original ZIP file.

7. Import the sandbox to Oracle Identity Manager.
8. Verify the changes and functionality of the new Home page tile.
9. Publish the sandbox to make the changes available to all users.

19.6.4 Showing Tiles Conditionally

By default, all the home page tiles are displayed. However, sometimes you might want to hide some of the tiles from certain users. For example, you might want to hide the Provisioning Tasks tile for end users. To do so, select the Grid Cell component that contains the Provisioning Tasks tile, and use the following EL for rendered property (shown as Show Component in Web Composer):

```
#{oimcontext.currentUser.adminRoles['OrclOIMSystemAdministrator'] != null}
```

See "Showing or Hiding UI Components Conditionally" on page 19-17 for details.

19.7 Developing Managed Beans and Task Flows

To implement advanced customization in Oracle Identity Manager, you can develop new task flows and managed beans by using JDeveloper IDE and then package them in the custom WAR file, which is `oracle.iam.ui.custom-dev-starter-pack.war`.

The beans are of the following types:

- **Request beans:** New instance of the bean is created for every request. JSFF component bindings and listeners are usually bound to request beans.
- **State beans:** Beans holding the state of the application, user session, or a particular flow. Values of components, such as af:inputText, can be bound to state beans. State beans must be serializable (implement java.io.Serializable) as ADF serializes/deserializes these beans between requests.

This section describes how to develop managed beans in the following topics:

- Setting Up the ViewController Project
- Setting Up a Model Project
- Adding Custom Managed Bean
- Deploying Custom Code to Oracle Identity Manager
- Using Managed Beans
- Using Managed Beans to Populate Request Attributes
- Using Public Taskflows
- Customizing Catalog Search
- Customizing Task Details Page for Approval Tasks

19.7.1 Setting Up the ViewController Project

Managed beans are created in a ViewController project. All your custom taskflows, pages, and managed beans must be present in the ViewController project.

To setup the ViewController project:

1. Create a new JDeveloper application. To do so:
   a. Start JDeveloper.

   b. Select **File**, **New**.
   c. Select **Generic Application**, and then click **OK**.
   d. Provide the application name and directory, and then click **Finish**. The application is created using a sample project.
   e. To delete the sample project, right-click the project, and select **Delete**.

2. Setup the ViewController project. To do so:
   a. Select **File**, **New**.
   b. Find and select **ADF ViewController Project**, and then click **OK**.
   c. Provide the project name, for example CustomUI, and project directory, and then click **Next**.
   d. Enter the default package name as **oracle.iam.ui.custom**, and then click **Finish**. The new project is created.
3. Add Oracle Identity Manager libraries to the project classpath. To do so:
   a. Right-click the new project, and select **Project Properties**.
   b. On the left navigation bar, select **Libraries and Classpath**.
   c. Click **Add Library**. Add ADF Model Runtime.
   d. Click **Add Library**, click **Load Dir**, provide the path as ***IDM_HOME*/server/jdev.lib**, and then click **OK**.
   e. From the list of libraries, select the following:
      – **OIM View Shared Library**
      – **OIM Model Shared Library**
      – **OIM Client Library**
   f. Click **OK**.
4. Define the deployment profile for the newly created ViewController project. To do so:
   a. Right-click the project, and select **Project Properties**.
   b. On the left navigation bar, select **Deployment**.
   c. Delete any existing deployment profiles.
   d. Click **New**, and select **ADF Library JAR File** as the archive type.

      Note: The ADF Library JAR File and JAR File archive types are different. Make sure that you select the **ADF Library JAR File** archive type.

   e. Provide and confirm the archive name, such as adflibCustomUI, and then click **OK**.

Your ViewController project setup is complete. You can now start adding custom taskflows, pages, and managed beans.

Note: Some examples in the consecutive sections in this document use the FacesUtils class. For information about this class, see Appendix A, "The FacesUtils Class".

19.7.2 Setting Up a Model Project

All your custom EOs/VOs and classes interacting directly with Oracle Identity Manager APIs must be present in a model project. To setup the model project:

1. Click **File**, **New**.
2. Find and select **ADF Model Project**, and then click **OK**.
3. Provide the Project Name, for example CustomModel, and Project Directory, and then click **Next**.
4. Enter Default Package name as **oracle.iam.ui.custom**, and then click **Finish**. The new project is created.
5. Add Oracle Identity Manager libraries to the project classpath:
   a. Right-click the project, and select **Project Properties**.
   b. On the left navigation bar, select **Libraries and Classpath**.
   c. Click **Add Library**.
   d. Click **Load Dir**, provide the path as ***IDM_HOME*/server/jdev.lib**, and then click **OK**.
   e. From the list of libraries select the following:
      – **OIM Model Shared Library**
      – **OIM Client Library**
   f. Click **OK**.
6. Define the deployment profile for the newly created model project. To do so:
   a. Right-click the project, and select **Project Properties**.
   b. On the left navigation bar, select **Deployment**.
   c. Delete any existing deployment profiles.
   d. Click **New**, and select **ADF Library JAR File** as the archive type.

   Note: The ADF Library JAR File and JAR File archive types are different. Make sure that you select the **ADF Library JAR File** archive type.

   e. Provide and confirm the archive name, such as adflibCustomModel, and then click **OK**.

Your model project setup is complete. You can now start adding custom EOs, VOs, and classes for interacting with Oracle Identity Manager APIs.

Note: Some examples in the consecutive sections in this document use the FacesUtils class. For information about this class, see Appendix A, "The FacesUtils Class".

19.7.3 Adding Custom Managed Bean

To add your custom managed bean:

1. Right-click the ViewController project, and select **New**.
2. Select the Java Class category.
3. Provide the class name, for example CustomReqBean or CustomStateBean, and the package name.
4. After creating the class, to register it with a taskflow:
    a. If you are developing your own bounded task flow, then navigate to your task flow definition file, and open it. Otherwise, locate the adfc-config.xml file in your ViewController project, and open it.
    b. Click the **Overview** tab, and select **Managed Beans**.
    c. Add a new managed bean entry. To do so:

       i) Provide managed bean name, for example customReqBean or customStateBean. This is the name that you will later use to refer to an instance of your bean.

       ii) Provide the managed bean class name.

       iii) Provide the scope. For request beans use backingBean scope. For state beans, use pageFlow scope.

Note:

- The pageFlow scope beans are visible only in the taskflow for which they are defined.
- To refer to your managed bean from JSFF/taskflow definition or other places, you can use EL expression. For example, if you register your bean under the name customReqBean and put the bean to backingBean scope, then you can reference your bean by using the following EL expression:

  ```
  #{backingBeanScope.customReqBean}
  ```

  If you put the bean to pageFlow scope, you can reference your bean by using the following EL expression:

  ```
  #{pageFlowScope.customStateBean}
  ```

19.7.4 Deploying Custom Code to Oracle Identity Manager

To deploy an ADF library JAR file produced by your custom model or ViewController projects:

1. Copy the oracle.iam.ui.custom-dev-starter-pack.war to a temporary location.
2. Open the oracle.iam.ui.custom-dev-starter-pack.war.
3. Add the custom jar file to the WEB-INF/lib directory. If the lib directory does not exist, then create it.
4. Save the oracle.iam.ui.custom-dev-starter-pack.war file.
5. Copy the oracle.iam.ui.custom-dev-starter-pack.war file back to its original location in the $*OIM_ORACLE_HOME*/server/apps/ directory.

6. Stop Oracle Identity Manager Managed Server.
7. In WebLogic Administration Console, update the oracle.iam.ui.custom library deployment, and activate the changes.
8. Start Oracle Identity Manager Managed Server.

19.7.5 Using Managed Beans

This section provides the following use cases for developing managed beans to customize Oracle Identity Manager interface:

- Showing Components Conditionally
- Prepopulating Fields Conditionally
- Setting a Conditional Mandatory Field
- Implementing Custom Field Validation
- Implementing Custom Cascading LOVs
- Customizing Forms By Using RequestFormContext
- Overriding the Submit Button in Request Catalog
- Launching Taskflows
- Creating an External Link

Note: The examples in this section use the FacesUtils class. For information about this class, see "The FacesUtils Class" on page A-1.

19.7.5.1 Showing Components Conditionally

You can show or hide certain fields conditionally based on the values of other fields. For example, to show the Contact Information panel on the Create User page only when the User Type is Full-Time Employee, perform the following steps:

1. In your custom request bean, define properties for component bindings of the User Type field and any parent component of the Contact Information panel, for example, the form root panel. To do so, use the following code:

```
private UIComponent rootPanelPGL;
    private UIComponent userTypeSOC;

    public void setRootPanelPGL(UIComponent rootPanelPGL) {
        this.rootPanelPGL = rootPanelPGL;
    }

    public UIComponent getRootPanelPGL() {
        return rootPanelPGL;
    }

    public void setUserTypeSOC(UIComponent userTypeSOC) {
        this.userTypeSOC = userTypeSOC;
    }

    public UIComponent getUserTypeSOC() {
        return userTypeSOC;
    }
```

2. Create or extend existing valueChangeListener that will be invoked when user selects the new value in the User Type list. To do so, use the following code:

   > **Note:** The listener will refresh the form.

```
public void valueChangeListener(ValueChangeEvent valueChangeEvent) {
        if (valueChangeEvent.getSource().equals(userTypeSOC)) {
            // refresh form
            FacesUtils.partialRender(rootPanelPGL);
        }
    }
```

3. Create a method that returns boolean value. The method will determine if the Contact Information panel is to be displayed when the page is rendered. In this example, the Contact Information panel will be shown if the User Type is Full-Time Employee.

   The method is as follows:

```
private static final String USER_TYPE_ATTRIBUTE = "usr_emp_type__c";

    public boolean isFullTimeEmployeeUserTypeSelected() {
        // return true if value of "usr_emp_type__c" binding attribute equals
to "Full-Time"
        // "usr_emp_type__c" binding attribute is used to display value of User
Type in the User Type drop-down
        return
"Full-Time".equals(FacesUtils.getListBindingValue(USER_TYPE_ATTRIBUTE,
String.class));
    }
```

4. Package and deploy the managed bean, as described in "Deploying Custom Code to Oracle Identity Manager" on page 19-37.

5. To bind the code with JSFF:

   a. Set component bindings for the User Type list and root panel components to point to the properties that you defined.

   b. Define the valueChangeListener for the User Type list.

   > **Note:** Make sure that the autosubmit property is set to true for the User Type list.

   c. Set EL expression for the rendered property, which is Show Component in Web Composer, on the Contact Information panel to point to the isFullTimeEmployeeUserTypeSelected() method defined in step 3.

   > **Note:** Ignore if the following error is displayed while setting EL expression for the rendered property:

```
"javax.faces.validator.ValidatorException:
java.lang.IllegalArgumentException: Control Binding
'usr_emp_type__c' not found"
```

19.7.5.2 Prepopulating Fields Conditionally

You prepopulate certain fields based on the values of other fields. For example, to prepopulate values in the User Login and E-mail fields on the Create User page based on the values of the First Name and Last Name fields, perform the following steps:

1. In your custom request bean, define properties for component bindings of First Name and Last Name fields and any parent component of the User Login and E-mail fields, for example, form root panel. To do so, use the following code:

```
private UIComponent firstNameIT;
    private UIComponent lastNameIT;
    private UIComponent rootPanelPGL;

    public void setFirstNameIT(UIComponent firstNameIT) {
        this.firstNameIT = firstNameIT;
    }

    public UIComponent getFirstNameIT() {
        return firstNameIT;
    }

    public void setLastNameIT(UIComponent lastNameIT) {
        this.lastNameIT = lastNameIT;
    }

    public UIComponent getLastNameIT() {
        return lastNameIT;
    }

    public void setRootPanelPGL(UIComponent rootPanelPGL) {
        this.rootPanelPGL = rootPanelPGL;
    }

    public UIComponent getRootPanelPGL() {
        return rootPanelPGL;
    }
```

2. Create or extend existing valueChangeListener that will be invoked when the user updates the First Name or Last Name fields. To do so, use the following code:

Note: The listener will update User Login and E-mail accordingly and refresh the form.

```
private static final String USER_LOGIN_ATTRIBUTE = "usr_login__c";
    private static final String EMAIL_ATTRIBUTE = "usr_email__c";
    private static final String LAST_NAME_ATTRIBUTE = "usr_last_name__c";
    private static final String FIRST_NAME_ATTRIBUTE = "usr_first_name__c";

    public void valueChangeListener(ValueChangeEvent valueChangeEvent) {
        if (valueChangeEvent.getSource().equals(firstNameIT)) {
            // get new value of first name from the event
           String firstName = (String)valueChangeEvent.getNewValue();
            // get existing value of last name through binding
            String lastName =
FacesUtils.getAttributeBindingValue(LAST_NAME_ATTRIBUTE, String.class);
            setUserLoginAndEmail(firstName, lastName);
        } else if (valueChangeEvent.getSource().equals(lastNameIT)) {
            // get existing value of first name through binding
            String firstName =
FacesUtils.getAttributeBindingValue(FIRST_NAME_ATTRIBUTE, String.class);
            // get new value of last name from the event
            String lastName = (String)valueChangeEvent.getNewValue();
            setUserLoginAndEmail(firstName, lastName);
        }
        // refresh form
        FacesUtils.partialRender(rootPanelPGL);
    }

    private void setUserLoginAndEmail(String firstName, String lastName) {
        StringBuilder sb = new StringBuilder();
        if (firstName != null) {
            sb.append(firstName);
        }
        if (firstName != null && !firstName.isEmpty() && lastName != null &&
!lastName.isEmpty()) {
            sb.append(".");
        }
        if (lastName != null) {
            sb.append(lastName);
        }
        String userLogin = sb.toString();
        // set new value for User Login and E-mail through binding
        FacesUtils.setAttributeBindingValue(USER_LOGIN_ATTRIBUTE, userLogin);
        FacesUtils.setAttributeBindingValue(EMAIL_ATTRIBUTE, userLogin +
"@acme.com");
}
```

3. Package and deploy the managed bean, as described in "Deploying Custom Code to Oracle Identity Manager" on page 19-37.
4. Add the code to the JSFF. To do so:
   a. Set the component bindings for First Name, Last Name, and root panel to point to the properties that you defined.
   b. Define valueChangeListener for First Name and Last Name input texts, and make sure that the autosubmit property is set to true on both input texts.

19.7.5.3 Setting a Conditional Mandatory Field

You can make a field conditionally mandatory based on the value of another field. For example, to make the Manager field on the Create User page mandatory only if the User Type is Intern, perform the following steps:

> **Note:** Enforcing field validation cannot be performed by setting the required property in Web Composer. You must develop a managed bean to perform field validation, as described in this section.

1. In your custom request bean, define properties for component bindings of the User Type field and any parent component of Manager field, for example, form root panel. To do so, use the following code:

```
private UIComponent rootPanelPGL;
    private UIComponent userTypeSOC;

    public void setRootPanelPGL(UIComponent rootPanelPGL) {
        this.rootPanelPGL = rootPanelPGL;
    }

    public UIComponent getRootPanelPGL() {
        return rootPanelPGL;
    }

    public void setUserTypeSOC(UIComponent userTypeSOC) {
        this.userTypeSOC = userTypeSOC;
    }

    public UIComponent getUserTypeSOC() {
        return userTypeSOC;
    }
```

2. Create or extend existing valueChangeListener that will be invoked when user selects new value in the User Type list. To do so, use the following code:

   Note: The listener will refresh the form.

```
public void valueChangeListener(ValueChangeEvent valueChangeEvent) {
        if (valueChangeEvent.getSource().equals(userTypeSOC)) {
            // refresh form
            FacesUtils.partialRender(rootPanelPGL);
        }
    }
```

3. Create a method that returns boolean value. The method determines whether or not the field is mandatory. In this example, the Manager field will be marked as mandatory if User Type is Intern.

   The method is as follows:

```
    public boolean isInternUserTypeSelected() {
        // return true if value of "usr_emp_type__c" binding attribute equals
to "Intern"
       // "usr_emp_type__c" binding attribute is used to display value of User
Type in the User Type drop-down
return
"Intern".equals(FacesUtils.getValueFromELExpression("#{bindings.usr_emp_type__c
.attributeValue}", String.class));
    }
```

4. Package and deploy the managed bean, as described in "Deploying Custom Code to Oracle Identity Manager" on page 19-37.

5. Add the code to the JSFF. To do so:

   a. Set component bindings for the User Type list and root panel components to point to the properties you defined.

   b. Define valueChangeListener for the User Type list. Make sure that the autosubmit property is set to true for the User Type list.

   c. Set EL expression for the required property on the Manager field to point to the isInternUserTypeSelected() method defined is step 3.

   d. Set EL expression for the Show required property on the Manager field panelLabelAndMessage to point to the isInternUserTypeSelected() method defined is step 3.

19.7.5.4 Implementing Custom Field Validation

Managed beans can be used to introduce custom validations. For example, you can implement the following validations for the Start Date and End Date fields on the Account Effective Dates panel of the Create User page:

- Start Date cannot be after End Date.
- The interval between Start Date and End Date cannot exceed 180 days for Contractors.

To implement custom validation using Managed Beans:

1. In your custom request bean, define properties for component bindings of the Start Date and End Date fields, as shown:

```
private UIComponent startDateID;
private UIComponent endDateID;

public void setStartDateID(UIComponent startDateID) {
    this.startDateID = startDateID;
}

public UIComponent getStartDateID() {
    return startDateID;
}

public void setEndDateID(UIComponent endDateID) {
    this.endDateID = endDateID;
}

public UIComponent getEndDateID() {
    return endDateID;
}
```

2. Add method for validation in your managed bean that will be invoked when the user selects new value for the Start Date or End Date field. The validator generates an error message when validation fails and attaches it to the field being updated. To do so, use the following code:

```
private static final String START_DATE_END_DATE_VALIDATION_MSG = "Start Date -
End Date interval cannot exceed 180 days for Contractors.";
    private static final String START_DATE_AFTER_END_DATE_VALIDATION_MSG =
"Start Date cannot be after End Date.";

    private static final String USER_TYPE_ATTRIBUTE = "usr_emp_type__c";
    private static final String START_DATE_ATTRIBUTE = "usr_start_date__c";
    private static final String END_DATE_ATTRIBUTE = "usr_end_date__c";

    public void validator(FacesContext facesContext, UIComponent uiComponent,
Object object) {
        if (uiComponent.equals(startDateID)) {
            // get value of End Date through binding
            oracle.jbo.domain.Date jboEndDate =
FacesUtils.getAttributeBindingValue(END_DATE_ATTRIBUTE,
oracle.jbo.domain.Date.class);
            // only validate if both Start Date and End Date are set
            if (jboEndDate != null) {
                // value of Start Date is passed to validator
                Date startDate = ((oracle.jbo.domain.Date)object).getValue();
                Date endDate = jboEndDate.getValue();
                validateStartDateEndDate(facesContext, uiComponent, startDate,
endDate);
            }
        } else if (uiComponent.equals(endDateID)) {
            // get value of Start Date through binding
            oracle.jbo.domain.Date jboStartDate =
FacesUtils.getAttributeBindingValue(START_DATE_ATTRIBUTE,
oracle.jbo.domain.Date.class);
            // only validate if both Start Date and End Date are set
            if (jboStartDate != null) {
                Date startDate = jboStartDate.getValue();
                // value of End Date is passed to validator
                Date endDate = ((oracle.jbo.domain.Date)object).getValue();
                validateStartDateEndDate(facesContext, uiComponent, startDate,
endDate);
            }
        }
    }

    private void validateStartDateEndDate(FacesContext facesContext,
UIComponent uiComponent, Date startDate, Date endDate) {
        Date startDatePlus180Days = new Date(startDate.getTime() + 180L * 24 *
60 * 60 * 1000);
        if (startDate.after(endDate)) {
            // queue error message for the component which is being validated
(either Start Date or End Date)
            facesContext.addMessage(uiComponent.getClientId(facesContext),
                                    new
FacesMessage(FacesMessage.SEVERITY_ERROR,
START_DATE_AFTER_END_DATE_VALIDATION_MSG, null));
        } else if (isContractorUserTypeSelected() &&
startDatePlus180Days.before(endDate)) {
            // queue error message for the component which is being validated
(either Start Date or End Date)
            facesContext.addMessage(uiComponent.getClientId(facesContext),
                                    new
FacesMessage(FacesMessage.SEVERITY_ERROR, START_DATE_END_DATE_VALIDATION_MSG,
null));
        } else {
            // re-render -- in case there was an error message in queue for any
of the two components it will be released
            FacesUtils.partialRender(startDateID);
            FacesUtils.partialRender(endDateID);
        }
    }

    public boolean isContractorUserTypeSelected() {
        // return true if value of "usr_emp_type__c" binding attribute equals
to "Contractor"
        // "usr_emp_type__c" binding attribute is used to display value of User
Type in the User Type drop-down
        return
"Contractor".equals(FacesUtils.getListBindingValue(USER_TYPE_ATTRIBUTE,
String.class));
    }
```

See Also: "The FacesUtils Class" on page A-1 for more information about the FacesUtils class

3. Package and deploy the managed bean, as described in "Deploying Custom Code to Oracle Identity Manager" on page 19-37.

4. Bind the code to the JSFF. To do so:

   a. Set component bindings for the Start Date and End Date fields to point to the properties that you defined.

   b. Define EL expression for validator property on Start Date and End Date fields to point to the validator method that you defined in step 2. For example:

```
<mds:attribute name="binding"
value="#{backingBeanScope.validatorBean.startDateID}"/>
      <mds:attribute name="validator"
value="#{backingBeanScope.validatorBean.validator}"/>
```

Note: The validator property cannot be added directly by using the Web Composer. This must be set manually in the MDS file for the JSFF. To do so:

1. Export the sandbox after setting component bindings for the Start Date and End Date fields by using the Web Composer.
2. Extract the contents of the ZIP file and locate the XML file for the form on which Start Date and End fields are modified. For example, the XML file for the Create User form is oracle/iam/ui/runtime/form/view/pages/mdssys/cust/site/site/user CreateForm.jsff.xml.
3. In a text editor, open the XML file and set validator for StartDate and EndDate fields. For Example:

```
<mds:modify element="_xg_36">
       <mds:attribute name="binding"
value="#{backingBeanScope.validatorBean.startDateID}"/>
      <mds:attribute name="validator"
value="#{backingBeanScope.validatorBean.validator}"/>
   </mds:modify>
   <mds:modify element="_xg_13">
      <mds:attribute name="binding"
value="#{backingBeanScope.validatorBean.endDateID}"/>
      <mds:attribute name="validator"
value="#{backingBeanScope.validatorBean.validator}"/>
   </mds:modify>
```

4. Save the changes, repackage the ZIP file (the sandbox archive), and then import it back to your environment.

19.7.5.5 Implementing Custom Cascading LOVs

Cascading LOVs are LOV components for which the list of values in one component is dependent on the currently selected value in another component. For example, based on the selected value in the User Type list on the Create User page, you might want to display the Job Code list or another LOV component whose list of values is dependent on the currently selected value in the User Type list.

The following are the high-level guidelines to implement custom cascading LOVs:

1. Define component binding for the User Type field and any parent component of Job Code, for example, form root panel.

2. Implement the model for Job Code LOV component by ensuring the following:

   - The model must take into account the current value of the User Type field.

- For af:selectOneChoice, you must implement a method that returns List<javax.faces.model.SelectItem>.
- For af:inputListOfValues, you must implement a method that returns an instance of oracle.adf.view.rich.model.ListOfValuesMode.

  See Also: "Using List-of-Values Components" in the *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework* for information about using a LOV component to display a model-driven list of objects from which a user can select a value

3. Implement valueChangeListener for the User Type field. Set the autosubmit property to true for the User Type field.

   valueChangeListener must update model of Job Code LOV component with the current value of the User Type field. In addition, valueChangeListener must re-render the form so that Job Code LOV component is updated with the current list of values.

19.7.5.6 Customizing Forms By Using RequestFormContext

RequestFormContext is a bean available in the pageFlowScope of entity form details taskflow. The entity forms include user form, application instance form, role form, and entitlement form. The instance provides various context information. Using this context information, you can customize various forms based on specific business requirements.

You can get an instance of the class by using Java code, as shown:

```
RequestFormContext.getCurrentInstance();
```

You can also get an instance of the class by using EL, as shown:

```
#{pageFlowScope.requestFormContext}
```

RequestFormContext provides the following context information:

- **operation:** The operation that is being performed on the entity. The possible values are CREATE and MODIFY.
- **actionType:** The action that is being performed by the user when the entity form is displayed. The possible values are: APPROVAL, FULFILL, REQUEST, VIEW, SUMMARY.
- **bulk:** Whether or not it is a bulk operation.
- **beneficiaryIds:** The list of beneficiary or target user IDs. For example, if you are requesting an application instance for the user John Doe, then the list contains the user ID of John Doe.
- **cartItemIds:** The list of cart item IDs. For example, if you are requesting an application instance for a user, then the list contains the application instance ID that is being requested.
- **requestEntityType:** The entity type being requested, which is any one of ROLE, ENTITLEMENT, APP_INSTANCE, USER.
- **requestEntitySubType:** The subtype of entity being requsted. For example, when requesting for an application instance, the requestEntitySubType is the application instance key.

- **instanceKey:** The key of the instance being modified.

The following is an example usage of the RequestFormContext:

You might want to add new Prepopulate button to the Create Application Instance form, and make the button visible only when there is only one target user. When the button is clicked, some of the application instance fields, such as User Login, First Name, and Last Name) will be prepopulated based on the current target user. To achieve this, perform the following steps:

1. In your custom request bean, define properties for component bindings of the Prepopulate button and the form root panel, as shown:

```
private UIComponent rootPanel;
    private UIComponent prepopulateButton;

    public void setRootPanel(UIComponent rootPanel) {
        this.rootPanel = rootPanel;
    }

    public UIComponent getRootPanel() {
        return rootPanel;
    }

    public void setPrepopulateButton(UIComponent prepopulateButton) {
        this.prepopulateButton = prepopulateButton;
    }

    public UIComponent getPrepopulateButton() {
        return prepopulateButton;
    }
```

2. Implement an actionListener that will be invoked when the Prepopulate button is clicked. The actionListener uses the target user ID and fetches user data, such as First Name and Last Name, by using Oracle Identity Manager API. Use the fetched data, and set certain application instance attributes through attribute binding, and finally refresh the form so that new values are displayed. The actionListener is as shown:

```
private static final String
ACCOUNT_LOGIN_ATTRIBUTE = "UD_EBS2722_LOGIN__c";
    private static final String ACCOUNT_ID_ATTRIBUTE =
"UD_EBS2722_ACCOUNTID__c";
    private static final String FIRST_NAME_ATTRIBUTE = "firstName__c";
    private static final String LAST_NAME_ATTRIBUTE = "lastName__c";
public void actionListener(ActionEvent e) {
        if (e.getSource().equals(prepopulateButton)) {
            RequestFormContext requestFormContext =
RequestFormContext.getCurrentInstance();
            List<String> beneficiaryIds =
requestFormContext.getBeneficiaryIds();
            if (beneficiaryIds.size() == 1) {
                // prepopulate fields based on selected beneficiary
                User user = getUser(beneficiaryIds.get(0));
                FacesUtils.setAttributeBindingValue(ACCOUNT_LOGIN_ATTRIBUTE,
user.getLogin());
                FacesUtils.setAttributeBindingValue(ACCOUNT_ID_ATTRIBUTE,
user.getId());
                FacesUtils.setAttributeBindingValue(FIRST_NAME_ATTRIBUTE,
user.getFirstName());
                FacesUtils.setAttributeBindingValue(LAST_NAME_ATTRIBUTE,
user.getLastName());
            }
        }
        FacesUtils.partialRender(rootPanel);
    }

    private User getUser(String userId) {
        UserManager userManager = OIMClientFactory.getUserManager();
        try {
            return userManager.getDetails(userId, null, false);
        } catch (NoSuchUserException e) {
            throw new RuntimeException(e);
        } catch (UserLookupException e) {
            throw new RuntimeException(e);
        }
    }
```

3. Create a method that returns Boolean value. The method determines if the Prepopulate button is to be displayed when the form is rendered. In this example, the Prepopulate button will be displayed when the number of target users is equal to 1. The method is as follows:

```
public boolean isPrepopulateButtonRendered() {
        RequestFormContext requestFormContext =
RequestFormContext.getCurrentInstance();
        return requestFormContext.getActionType() ==
RequestFormContext.ActionType.REQUEST &&
requestFormContext.getBeneficiaryIds().size() == 1;
    }
```

4. Package and deploy the managed bean. See "Deploying Custom Code to Oracle Identity Manager" on page 19-37 for information.

5. Bind the code with JSFF. To do so:

   a. Add a Prepopulate button to the Create Application Instance form.

   b. Set bindings for the Prepopulate button and the root panel.

   c. Set the Prepopulate button actionListener property to point to the actionListener method implemented in step 2.

   Note: The actionListener property cannot be set by using the Web Composer. This must be set manually as follows:

   1. Export the sandbox.
   2. Edit the JSFF to set the actionListener attribute value. For example:

   ```
   <mds:attribute name="actionListener"
   value="#{backingBeanScope.accountFormReqBean.submitButtonAction
   Listener}"/>)
   ```

   3. Import the updated sandbox.

   This procedure is applicable to setting the actionListener property in all the examples in this document.

   d. Set the rendered property to point to the isPrepopulateButtonRendered() method implemented in step 3.

19.7.5.7 Overriding the Submit Button in Request Catalog

Can override the Submit button in the request catalog and execute additional logic based on your requirements. For example, to add additional check for number of target users or beneficiaries when submitting a request, and allow submitting the request when the number of beneficiaries is not more than five, perform the following steps:

1. Implement actionListener that will override the original Submit button.

   The actionListener will be invoked when the user clicks the Submit button. The actionListener performs the extra check and either display error messages or executes the original actionListener bound to the Submit button. Original Submit button actionListener can be executed using the following EL expression:

   ```
   #{backingBeanScope.cartReqBean.submitActionListener}
   ```

   The actionListener code is as shown:

   ```
   private static final String MORE_THAN_FIVE_TARGET_USERS_MSG = "Cannot submit
   request for more than five target users.";
   public void submitButtonActionListener(ActionEvent e) {
           // only submit request if there is no more than 5 beneficiaries
           Boolean moreThanFiveTargetUsers =
   FacesUtils.getValueFromELExpression("#{backingBeanScope.cartReqBean.targetUserS
   ize > 5}", Boolean.class);
           if (moreThanFiveTargetUsers) {
               // display error message
               FacesMessage fm = new FacesMessage();
               fm.setSeverity(FacesMessage.SEVERITY_ERROR);
               fm.setSummary(MORE_THAN_FIVE_TARGET_USERS_MSG);
               FacesUtils.showFacesMessage(fm);
           } else {
               // execute original submit button action listener
               MethodExpression originalActionListener =

   FacesUtils.getMethodExpressionFromEL("#{backingBeanScope.cartReqBean.submitActi
   onListener}", null, new Class[] { ActionEvent.class });
               originalActionListener.invoke(FacesUtils.getELContext(), new
   Object[] { e });
           }
       }
   ```

2. Update the Submit button actionListener property to point to the new actionListener implementation.

19.7.5.8 Launching Taskflows

You can launch a taskflow in the Self Service interface. For example, if you want to launch a tab with a bounded taskflow running in it, then perform the following steps:

1. Develop a custom managed bean with the following method, which is also called action listener:

   ```
   public void launchMyTaskFlow(ActionEvent evt){

   User user =
   OIMClientFactory.getAuthenticatedSelfService().getProfileDetails(null);
          String taskFlowId =
   "/WEB-INF/oracle/iam/ui/taskflows/public/tfs/user-details-tf.xml#user-details-t
   f";
          // This id uniquely identifies the taskflow after launch. Add a suffix,
   ```

```
for example entityPrimaryKey, to make it unique.
       String id = "user-detail-tf";
       String name = user.getDisplayName() ;  // this is shown as the tab title
       String description = ""; // Add any suitable description
       String icon = "/images/user.png";
       String  helpTopicId = ConstantsDefinition.DEFAULT_HELP_TOPIC_ID; // Or
your custom OHW integrated help topic id
       boolean inDialog = false;
       Map params = new HashMap();  // These are your taskflow's input
parameters being passed from this launcher method
       params.put("userLogin",  user.getLogin());

       String jsonPayLoad = TaskFlowUtils.createContextualEventPayLoad(id,
taskFlowId, name, icon, description, helpTopicId, inDialog, params);

TaskFlowUtils.raiseContextualEvent(TaskFlowUtils.RAISE_TASK_FLOW_LAUNCH_EVENT,
jsonPayLoad);
 }
```

Note: The above code snippet uses the user details public taskflow to display the user details when the user login is provided. For a list of available public taskflows that you can use for customization of UI, see "Using Public Taskflows" on page 19-57.

Package and deploy the managed bean, as described in "Deploying Custom Code to Oracle Identity Manager" on page 19-37.

2. Using sandbox and Web Composer customization, add an ADF CommandLink to the correct page (JSFF file). Open the sandbox zip, and edit the jsff.xml to bind actionListener for that link to the managed bean method.

3. Ensure that the page definition of the jsff has the raiseTaskFlowLaunchEvent binding. To find the name of the page definition file, you first need to know the name of the jsff page on which you have the launch link.

   If your launch link is on a custom jsff page, for example, your page name is my-custom.jsff, then look for a file named my-custom_pageDef.xml within the same JDev project. JDev automatically creates this file for each jsff. You must add the following eventBinding into this pageDef xml file:

```
<eventBinding id="raiseTaskFlowLaunchEvent">
      <events xmlns="http://xmlns.oracle.com/adfm/contextualEvent">
        <event name="oracle.idm.shell.event.TaskFlowLaunchEvent"/>
      </events>
    </eventBinding>
```

Note: Existing Oracle Identity Manager pages already contain the eventBinding. You must define the eventBinding for JSFF pages that you build.

Oracle Identity Manager allows you to add your own UI or taskflows, such as goLink, commandLink, commandButton, or launch a taskflow. Perform the following steps to add your custom UI or taskflow:

1. Write a managed bean and register using adfc-config.xml in oracle.iam.ui.custom-dev-starter-pack.war.
2. Add a new commandLink or commandButton on the page where you want to display the link or button by using Web Composer.
3. Set the actionListener property of the link or button component that you added to point to the actionListener method.
4. Raise the contextual event using the managed bean, which will be handled by Oracle Identity Manager. The taskflow is launched.

19.7.5.9 Creating an External Link

To add a link or button that redirects the user to a certain URL:

1. In your custom request bean, create the following actionListener that will be invoked when the user clicks a link or button:

   ```
   public void actionListener(ActionEvent e) {
           FacesUtils.redirect("http://www.oracle.com");
       }
   ```

2. Package and deploy the managed bean. See "Deploying Custom Code to Oracle Identity Manager" on page 19-37 for information.
3. Add a new commandLink or commandButton to the page on which you want to display the link or button by using Web Composer. See "Adding a Link or Button" on page 19-20 for details.
4. Set the actionListener property of the link or button component that you added to point to the actionListener method.

19.7.6 Using Managed Beans to Populate Request Attributes

This section describes the following approaches for populating request attributes:

- Populating Request Attributes Using Managed Beans
- Populating Request Attributes by Using the Prepopulate Plug-in

19.7.6.1 Populating Request Attributes Using Managed Beans

This approach involves creating a managed bean that gets invoked when the user clicks a custom button. The managed bean must be deployed to the Oracle Identity Manager customization placeholder library, which is oracle.iam.ui.custom-dev-starter-pack.war. The button, referred to as the Prepopulate button, is part of the UI customization and must be manually added to the page by using Web Composer.

The managed bean code is responsible for fetching the information to be populated in the request form. It uses Oracle Identity Manager APIs to get the beneficiary information from the request and from the user management layer, and uses JSF/ADF APIs to update the request form UI components.

To populate request attributes by using managed beans and UI customization:

1. Create the JDev application workspace and project. See "Setting Up the ViewController Project" on page 19-34 for details.
2. Create a Java class. In this example, the complete class name is com.oracle.demo.iam.prepop.view.PrePopulateMBean. This class contains:

- Two member variables that hold references to the UI components, the custom Prepopulate button and its parent container.
- Accessor methods (get and set) for the variables member variables.
- An action listener type method to be invoked when the user clicks the custom Prepopulate button.
- A method that returns a boolean value determines when the custom Prepopulate button must be disabled

The custom code for this example is:

```
public class PrePopulateMBean {

    private UIComponent rootPanel;
    private UIComponent prepopulateButton;

    public PrePopulateMBean() {
        super();
    }

    public void setRootPanel(UIComponent rootPanel) {
        this.rootPanel = rootPanel;
    }

    public UIComponent getRootPanel() {
        return rootPanel;
    }

    public void setPrepopulateButton(UIComponent prepopulateButton) {
        this.prepopulateButton = prepopulateButton;
    }

    public UIComponent getPrepopulateButton() {
        return prepopulateButton;
    }

    public boolean isPrepopulateButtonRendered() {

        boolean ret = false;
        RequestFormContext requestFormContext =
RequestFormContext.getCurrentInstance();
        if (requestFormContext != null) {

            boolean isActionRequest   = (requestFormContext.getActionType() ==
RequestFormContext.ActionType.REQUEST);
            boolean singleUserRequest = false;

            if (requestFormContext.getBeneficiaryIds()!=null) {
                singleUserRequest =
(requestFormContext.getBeneficiaryIds().size() == 1);
            }
            ret = isActionRequest && singleUserRequest;
        }
        return (ret);
    }

    public void actionListener(ActionEvent e) {

        if (e.getSource().equals(prepopulateButton)) {
            RequestFormContext requestFormContext =
RequestFormContext.getCurrentInstance();
            List<String> beneficiaryIds =
requestFormContext.getBeneficiaryIds();

            if (beneficiaryIds.size() == 1) {

                try {
                    User user = getUser(beneficiaryIds.get(0));
                    FacesUtils.setAttributeBindingValue("UD_OID_USR_FNAME__c",
user.getFirstName());
                    FacesUtils.setAttributeBindingValue("UD_OID_USR_LNAME__c",
user.getLastName());

                } catch (NoSuchUserException f) {
                    f.printStackTrace();
                } catch (UserLookupException f) {
                    f.printStackTrace();
                }
            }
        }
        FacesUtils.partialRender(rootPanel);
    }

    private User getUser(String userKey) throws NoSuchUserException,
UserLookupException {

        UserManager userMgr = OIMClientFactory.getUserManager();

        HashSet<String> searchAttrs = new java.util.HashSet<String>();
        searchAttrs.add(AttributeName.USER_LOGIN.getId());
        searchAttrs.add(AttributeName.LASTNAME.getId());
        searchAttrs.add(AttributeName.FIRSTNAME.getId());

        return userMgr.getDetails(userKey,searchAttrs, false);
    }
}
```

In the code for the Java class:

- The `isPrepopulateButtonRendered` method returns `true` if a RequestContext is available, and if there is only one request beneficiary. The check on the RequestContext availability is required to avoid issues at the time of customization. This method is invoked when the custom Prepopulate button is loaded, or its container is refreshed.
- The `actionListener` method executes a user search in Oracle Identity Manager by invoking the `getUser` method, which uses the request beneficiary information. Then, it directly sets values on the UD_OID_USR_FNAME__c and UD_OID_USR_LNAME__c UI components with the information returned from the user search, and invokes a partial rendering on the rootPanel. This is the panel that holds the custom button and the request form. The partial rendering will display the values in the respective fields. It is important to mention here that this custom code contains a direct reference to the UI components, and that these direct references can be found by exporting the sandbox. This method is invoked when the custom Prepopulate button is loaded or its container refreshed.
- The FacesUtil class is responsible for rendering the UI changes. See Appendix A, "The FacesUtils Class" for the code for this class.

3. Declare the PrePopulateMBean class as a managed bean in the JDev project. This makes the MBean available in the UI so that it can be invoked by using EL expressions. To configure this, specify the following values in the Managed Beans section of the View Controller project:

   - **Name:** prepopMBean
   - **Class:** com.oracle.demo.iam.prepop.view.PrePopulateMBean
   - **Scope:** backingBean

4. Deploy the View Controller project as an ADF library JAR file. This type of deployment can be created in JDeveloper through the deployment profiles option. The deployment generates a JAR file. Copy this file into oracle.iam.ui.custom-dev-starter-pack.war, which is Oracle Identity Manager placeholder library. This file is available along with the other Oracle Identity Manager application packages, such as EAR and WAR files, at the $*OIM_ORACLE_HOME*/server/apps/ directory. Create a backup of this file before modifying it.

5. Deploy the custom code. See "Deploying Custom Code to Oracle Identity Manager" on page 19-37 for information.

6. Customize the UI. To do so:

   a. In Oracle Identity Self Service, create and activate a sandbox. In this example, the sandbox name is `RequestPrePop`.

   b. Navigate to the access catalog.

   c. Search for the specific application instance to be customized. In this example, the application instance is called `Local OID`. Add the application instance to the cart, and click **Checkout**.

   d. Click **Customize**.

   e. Select **Structure** to open the component tree.

   f. In the Cart Items and Details sections of the page, click close to the Details label. Make sure that the showDetailHeader:Details component is selected.

   g. Click **Edit**. In the dialog box that opens, edit the Binding property, and configure the following EL using the Expression Builder:

   ```
   #{backingBeanScope.prepopMBean.rootPanel}
   ```

   This expression bind will make the UI invoke the setRootPanel method in the custom managed bean. Click **OK**.

   h. Make sure that the showDetailHeader:Details component is selected. Click **Add Content**.

   i. Scroll down, and open the Web Components section.

   j. Click **Add** on the right of the Command Toolbar Button component. A button is added on the Details section.

   k. Click the button, and then click **Edit**.

   l. Edit the Text property, and set PrePopulate as the label.

   m. Edit the Binding property and configure the following EL using the Expression Builder:

   ```
   #{backingBeanScope.prepopMBean.prepopulateButton}
   ```

This bind is for invoking the setPprepopulateButton method in the custom managed bean. Click **OK**.

n. Edit the Disabled property, and configure the following EL by using the Expression Builder:

```
#{!backingBeanScope.prepopMBean.prepopulateButtonRendered}
```

This is to invoke the isPrepopulateButtonRendered method in the managed bean. Click **Ok**.

o. Click the **Style** tab. Set the Width property to 100, and the Margin - Left property to 100. Click **OK**. This configuration will properly place the PrePopulate button in the UI.

p. Exit the customization mode by clicking **Close**.

7. To manually configure the properties of the Prepopulate button:

a. Navigate to the Sandbox page. De-activate and export the sandbox.

b. Save the sandbox ZIP file in the local file system.

c. Extract the ZIP file. In a text editor, open the XML file corresponding to the customization. In this example, the file is oracle/iam/ui/runtime/form/view/pages/mdssys/cust/site/site/OIDUserFormCreateForm.jsff.xml.

d. Search for the section defining the custom Prepopulate button, which can be similar to the following:

```
<af:commandToolbarButton xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
id="e8829502064"
binding="#{backingBeanScope.prepopMBean.prepopulateButton}"
text="PrePopulate"
```

e. Add the actionListener property to the custom Prepopulate button, as shown:

```
actionListener="#{backingBeanScope.prepopMBean.actionListener}
```

f. Save the file and repackage the ZIP. Make sure that the path is preserved when repacking the contens.

g. Import the sandbox, and import the ZIP file. Make sure that the sandbox is not active when importing it.

h. Activate the sandbox.

8. Test the customization. To do so:

a. Navigate to the Catalog, find the application instance and add it do the shopping cart.

b. In the cart summary page, the custom Prepopulate button is displayed.when clicking on it, the First Name and Last Name fields will be updated with the beneficiary's information

c. Click the Prepopulate button. The First Name and Last Name fields are updated with the beneficiary's information.

9. Publish the sandbox.

19.7.6.2 Populating Request Attributes by Using the Prepopulate Plug-in

Prepopulate plug-ins can be used when the same logic is to be executed for both UI and API request creation, and can also be used when a UI interaction is not required. In this approach, a plug-in is present for each attribute that must be prepopulated in the request. The same plug-in can be used across different resources and different attributes.

The plug-in code implements the oracle.iam.request.plugins.PrePopulationAdapter interface. The following is an example code:

```
package com.oracle.demo.iam.prepop.plugin;

import java.io.Serializable;

import java.util.HashSet;
import java.util.List;

import oracle.iam.identity.usermgmt.api.UserManager;
import oracle.iam.identity.usermgmt.api.UserManagerConstants.AttributeName;
import oracle.iam.identity.usermgmt.vo.User;
import oracle.iam.platform.Platform;
import oracle.iam.request.vo.Beneficiary;
import oracle.iam.request.vo.RequestData;

public class UserLoginPrePop implements
oracle.iam.request.plugins.PrePopulationAdapter {

    public UserLoginPrePop() {
        super();
    }

    public Serializable prepopulate(RequestData requestData) {

        String prePopUserId = null;

        List<Beneficiary> benList = requestData.getBeneficiaries();

        if(benList.size()==1){

            UserManager  usersvc = Platform.getService(UserManager.class);

            for(Beneficiary benf: benList){

                HashSet<String> searchAttrs = new java.util.HashSet<String>();
                searchAttrs.add(AttributeName.USER_LOGIN.getId());

                try {
                    User userBenef =
usersvc.getDetails(benf.getBeneficiaryKey(),searchAttrs, false);
                    if (userBenef!= null) {
                        prePopUserId = userBenef.getLogin();
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
        return prePopUserId;
    }
}
```

A prepopulate plug-in is similar to any other plug-in in Oracle Identity Manager. The plug-in class is compiled and deployed to a JAR file. The JAR file must be added to a ZIP file in the lib directory. The ZIP file must contain in the root path a XML file declaring the plug-in. The XML used in this example is as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<oimplugins xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <plugins pluginpoint="oracle.iam.request.plugins.PrePopulationAdapter">
 <plugin pluginclass= "com.oracle.demo.iam.prepop.plugin.UserLoginPrePop"
version="1.0" name="UserLoginPrePop">
  <metadata name="PrePopulationAdapater">
   <value>OracleDBUMForm::Username|OIDUserForm::User ID</value>
  </metadata>
 </plugin>
</plugins>
</oimplugins>
```

In the XML code:

- The xmlns tag attribute must be present in the XML. Otherwise, the plug-in is not invoked by Oracle Identity Manager.
- The value in the pluginpoint element must be oracle.iam.request.plugins.PrePopulationAdapter.
- The metadata tag contains a value child node. This value child node must contain the pairs of FormName::AttributeName. Each pair indicates a form attribute that will be populated by the prepopulate plug-in. In this example, such attributes are Username in the OracleDBUMForm form and User ID in the OIDUserForm form. The form names are configured when the ApplicationInstances and their forms were created, and not the process form created when the connector is imported into Oracle Identity Manager.

The prepopulate plug-in can be deployed to the $*OIM_HOME*/server/plugins/ directory, or it can be registered using the plug-in registration script. In production environments, it is always recommended to deploy the plug-in by using the command line so that the plug-in Zip file is uploaded to the database.

19.7.7 Using Public Taskflows

Oracle Identity Manager provides default taskflows for using them in the customized pages of Oracle Identity Self Service and to invoke other taskflows. For example, you can customize the user details page so that the user details of the manager will be displayed if you click the manager login name in the user details page.

The default or predefined taskflows are called public taskflows. While launching the public taskflows, you must provide appropriate values for some parameters. For example, to launch the request details page for a particular request, you must provide the request ID for the request.

Table 19–6 lists the public taskflows provided by Oracle Identity Manager along with the input parameters that are required to invoke the taskflows.

Table 19–6 Public Taskflows

Taskflow name	Taskflow path	Description	Parameter	Mandatory
Request Details	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/request-details-tf.xml#request-details-tf	This is launched to view the details of a request that is submitted for approval.	**requestID:** The ID of the request whose details is to be displayed.	Yes
User Details	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/user-details-tf.xml#user-details-tf	This is launched to view the details of a user.	**userLogin:** User Login attribute value of the user whose details is to be displayed.	Yes
Role Details	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/role-details-tf.xml#role-details-tf	This is launched to view the details of a role.	**roleName:** Name of the role whose details is to be displayed.	Yes
Request Role	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/request-role-tf.xml#request-role-tf	This is launched to request for assignment of role(s) for beneficiaries.	**roleNames:** Names of the role(s) that are to be assinged. The names must be separated by commas.	Yes
			userLogins: User Login attribute values of the user(s) or beneficaries for whom the roles are to be assigned. The values must be separated by commas. If a value is not provided, then the request action is applicable for the currently logged-in user.	No
Revoke Role	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/revoke-role-tf.xml#revoke-role-tf	This is launched to request for revoking of role(s) that are assigned to beneficiaries.	**roleNames:** Names of the role(s) that are to be revoked.	Yes
			userLogins: User Login attribute values of the user(s) or beneficiaries for whom the roles are to be revoked. The values must be separated by commas. If a value is not provided, then the revoke action is applicable for the currently logged-in user.	No

Table 19–6 (Cont.) Public Taskflows

Taskflow name	Taskflow path	Description	Parameter	Mandatory
Request Account	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/request-account-tf.xml#request-account-tf	This is launched to request for creation of account(s) for the beneficiaries.	**appInstNames:** Names of the application instance(s) where accounts are to be created. The values must be separated by commas.	Yes
			userLogins: User Login attribute values of the user(s) or beneficiaries for whom the accounts are to be assigned. The values must be separated by commas. If a value is not provided, then the request action is applicable for the currently logged-in user.	No
Modify Account	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/modify-account-tf.xml#modify-account-tf	This is launched to modify the account details created for a user or beneficiary.	**accountNames:** Name of the accounts whose details are to be modified. The values must be separated by commas.	Yes
			userLogins: User Login attribute values of the users or beneficiaries whose account details are to be modified. The values must be separated by commas. If a value is not provided, then the revoke action is applicable for the currently logged-in user.	No
Enable Account	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/enable-account-tf.xml#enable-account-tf	This is launched to enable accounts assigned to user(s) or beneficiaries.	**accountNames:** Names of the accounts that are to be enabled. The values must be separated by commas.	Yes

Table 19–6 (Cont.) Public Taskflows

Taskflow name	Taskflow path	Description	Parameter	Mandatory
			userLogins: User Login attribute values of the user(s) or beneficiaries whose accounts are to be enabled. The values must be separated by commas. If a value is not provided, then the request action is applicable for the currently logged-in user.	No
Disable Account	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/disable-account-tf.xml#disable-account-tf	This is launched to disable accounts assigned to user(s) or beneficiaries.	**accountNames:** Names of the accounts that are to be disabled. The values must be separated by commas.	Yes
			userLogins: User Login attribute values of the user(s) or beneficiaries whose accounts are to be disabled. The values must be separated by commas. If a value is not provided, then the request action is applicable for the currently logged-in user.	No
Delete Account	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/delete-account-tf.xml#delete-account-tf	This is launched to delete accounts assigned to user(s) or beneficiaries.	**accountNames:** Names of the accounts that are to be deleted. The values must be separated by commas.	Yes
			userLogins: User Login attribute values of the user(s) or beneficiaries whose accounts are to be deleted. The values must be separated by commas. If a value is not provided, then the request action is applicable for the currently logged-in user.	No
Request Entitlement	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/request-entitlement-tf.xml#request-entitlement-tf	This is launched to request for the assignment of entilement(s) for beneficiaries.	**entlmntNames:** Names of the entilement(s) that are to be assigned. The values must be separated by commas.	Yes

Table 19–6 (Cont.) Public Taskflows

Taskflow name	Taskflow path	Description	Parameter	Mandatory
			userLogins: User Login attribute values of the user(s) or beneficiaries to whom the entitlements are to be assigned. The values must be separated by commas. If a value is not provided, then the request action is applicable for the currently logged-in user.	No
Revoke Entitlement	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/revoke-entitlement-tf.xml#revoke-entitlement-tf	This is launched to request for revoking of entilement(s) assigned to beneficiaries.	**entlmntNames:** Names of the entilement(s) that are to be revoked. The values must be separated by commas.	Yes
			userLogins: User Login attribute values of the user(s) or beneficiaries from whom the entitlements are to be revoked. The values must be separated by commas. If a value is not provided, then the request action is applicable for the currently logged-in user.	No
Create User	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/create-user-tf.xml#create-user-tf	This is launched to create an user entity.	No parameters are required.	No
Modify User	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/modify-user-tf.xml#modify-user-tf	This is launched to modify the user details.	**userLogins:** User Login attribute values of the user(s) whose details are to be modified. If more than one userLogin attribute is provided as parameter, then bulk modify page is displayed. The values must be separated by commas.	Yes
Enable User	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/enable-user-tf.xml#enable-user-tf	This is launched to enable the disabled user(s).	**userLogins:** The User Login attribute values of the users that are to be enabled. The values must be separated by commas.	Yes

Table 19–6 (Cont.) Public Taskflows

Taskflow name	Taskflow path	Description	Parameter	Mandatory
Disable User	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/disable-user-tf.xml#disable-user-tf	This is launched to disable the enabled user(s).	**userLogins:** The User Login attribute values of the users that are to be disabled. The values must be separated by commas.	Yes
Delete User	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/modify-user-tf.xml#modify-user-tf	This is launched to delete user(s).	**userLogins:** The User Login attribute values of the users that are to be deleted. The values must be separated by commas.	Yes

Table 19–6 (Cont.) Public Taskflows

Taskflow name	Taskflow path	Description	Parameter	Mandatory

Catalog Search	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/catalog-search-tf.xml#catalog-search-tf	This is launched to specify the catalog search criteria and display the search results page or cart details page directly without using the catalog search page.	**searchCrtieria:** Various string attributes in the following format:
			`{criteriaName: "string", allowSearch: "true/false", profileName: "string",`
			
			`directCheckout: "true/false", showEntityTypeSelector: "true/false",`
			
			`hiddenTag: "string", allowedEntityTypes: "string", tags: "string",`
			
			`entityType: "string", auditObjective: "string", riskLevel: "string",`
			
			`ANY_UDF: "string"}`
			
			Here:
			■ **criteriaName:** Optional string attribute that will be displayed in the catalog results page.
			■ **allowSearch:** Optional boolean attribute to control rendering of tag search field in results page.
			■ **profileName:** Optional string attribute to take user to cart page by simulating the saved profile click.
			■ **directCheckout:** Optional parameter to add search results to the cart and take user to checkout page (true/false).
			■ **showEntityTypeSelector**Optional boolean attribute to show entityTypeSelector dropdown. This is displayed only if allowSearch is also set to true
				Yes

Table 19–6 (Cont.) Public Taskflows

Taskflow name	Taskflow path	Description	Parameter	Mandatory
			userLogins: The User Login attribute values of the users to be displayed in the beneficiary table in the catalog search results page. The values must be separated by commas. If value is not passed, then the current logged-in user is shown in the beneficiary table.	
Catalog Item Details	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/catalog-item-details-tf.xml#catalog-item-details-tf	This is launched to display the details of a catalog item.	**catalogItemName:** Name of the catalog item whose details are to be displayed.	Yes
			catalogItemType: Type of the catalog item whose details are to be displayed. The valid values are Role, ApplicationIstance, or Entitlement.	Yes
User Roles	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/user-roles-tf.xml#user-roles-tf	This is launched to view the roles page of a given user.	**userLogin:** User Login attribute value of the user whose roles page is to be displayed.	Yes
User Accounts	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/user-accounts-tf.xml#user-accounts-tf	This is launched to view the accounts page of a given user.	**userLogin:** User Login attribute value of the user whose accounts page is to be displayed.	Yes
User Entitlements	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/user-entitlements-tf.xml#user-entitlements-tf	This is launched to view the entitlements page of a given user.	**userLogin:** User Login attribute value of the user whose entitlements page is to be displayed.	Yes
User Assigned Admin Roles	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/user-assigned-adminroles-tf.xml#user-assigned-adminroles-tf	This is launched to view the assigned admin roles page of a given user.	**userLogin:** User Login attribute value of the user whose assigned admin roles page is to be displayed.	Yes
Organization Details	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/org-details-tf.xml#org-details-tf	This is launched to view the organization details page.	**orgName:** Name of the organization whose details page is to be displayed.	Yes
My Access	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/my-access-tf.xml#my-access-tf	This is launched to display the access page of the currently logged-in user.	No parameters are required.	No
Change User Account Password	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/account-passwd-reset-tf.xml#account-passwd-reset-tf	This is launched to display the change user account password page for a given user.	**accountName:** Name of the user's account whose password is to be changed.	Yes

Table 19–6 (Cont.) Public Taskflows

Taskflow name	Taskflow path	Description	Parameter	Mandatory
			userLogin: User Login attribute value of the user whose account password is to be changed.	Yes
Account Details	/WEB-INF/oracle/iam/ui/taskflows/public/tfs/account-details-tf.xml#account-details-tf	This is launched to display the details of a user's account.	**accountName:** Name of the user's account whose details is to be displayed.	Yes
			userLogin: User Login attribute value of the user whose account is to be displayed.	Yes

Note:

- The parameters of all the public taskflows listed in Table 19–6 are of type java.lang.String.
- The public taskflows can be launched by using contextual event as described in "Launching Taskflows" on page 19-49. Otherwise, public taskflows can be embedded in an ADF faces page. To embed public taskflows in an ADF faces page, the following parameter (in addition to the parameters listed in Table 19–6) must be added to the taskflow definition in the page definition file of the ADF faces page:

  Parameter name: "uiShell"

  value: "#{pageFlowScope.uiShell}"

19.7.8 Customizing Catalog Search

For customizing the default catalog search form, see "Configuring the Access Request Catalog" in *Administering Oracle Identity Manager*.

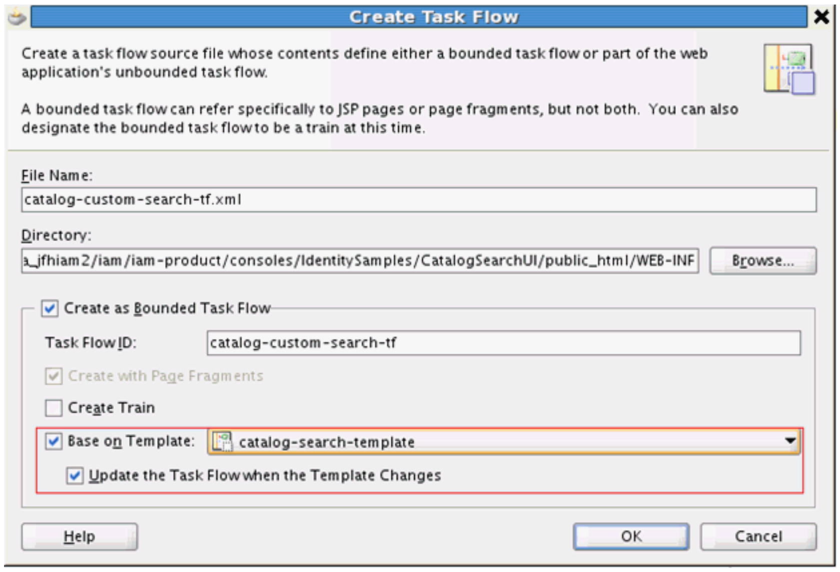
For advanced customizations to the catalog search, such as adding search fields and search operators, it is recommended to create a custom taskflow and then replace the default catalog taskflow with the custom taskflow.

To implement a custom taskflow for catalog search:

1. Develop the custom taskflow as a bounded taskflow based on page fragments in the ViewController project. Make sure that OIM client, OIM model, and OIM view libraries are also added to the ViewController project. For information about setting up the ViewController project, see "Setting Up the ViewController Project" on page 19-34.

   The custom taskflow can be based on the taskflow template `/WEB-INF/oracle/iam/ui/catalog/tfs/catalog-search-template.xml`, as shown in Figure 19–7:

Figure 19–7   Catalog Taskflow Based on Template



Create a task flow source file whose contents define either a bounded task flow or part of the web application's unbounded task flow.

A bounded task flow can refer specifically to JSP pages or page fragments, but not both. You can also designate the bounded task flow to be a train at this time.

File Name: catalog-custom-search-tf.xml

Directory: a_jfhiam2/iam/iam-product/consoles/IdentitySamples/CatalogSearchUI/public_html/WEB-INF

☑ Create as Bounded Task Flow

Task Flow ID: catalog-custom-search-tf

☑ Create with Page Fragments

☐ Create Train

☑ Base on Template: catalog-search-template

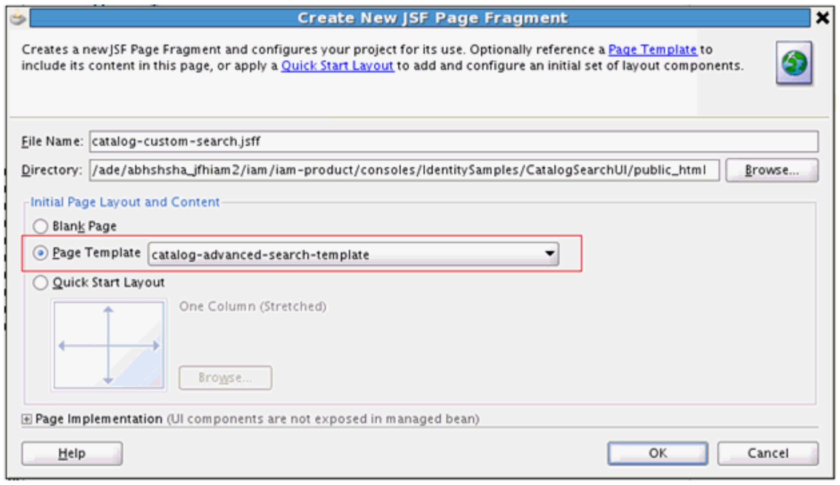
☑ Update the Task Flow when the Template Changes

By extending this template, values to the following parameters are automatically passed to the custom taskflow. These parameters can be accessed from `pageFlowScope` in the custom taskflow.

- **entityType:** When requesting for roles, entitlements, or accounts from the My Access page of Identity Self Service, the value passed to this parameter is Role, Entitlement, Application Instance respectively.
- **showEntityTypeSelector:** When requesting for roles, entitlements, or accounts from the My Access page, the value for this parameter is passed as false. This parameter can be used to hide the entity type selector in the custom taskflow.
- **showAppSelector:** When requesting for entitlements from the My Accounts tab in the My Access page, the value for this parameter is passed as false. This parameter can be used to hide the application selector in the custom taskflow.
- **parentEntityKey:** When requesting for entitlement from the My Accounts tab in the My Access page, the application instance key corresponding to the selected account is passed to this parameter.

2. The page fragment in the custom taskflow can be based on the template `/oracle/iam/ui/catalog/pages/catalog-advanced-search-template.jspx`, as shown in Figure 19–8:

Figure 19–8 JSF Page Fragment Based on Page Template


Create New JSF Page Fragment

Creates a new JSF Page Fragment and configures your project for its use. Optionally reference a Page Template to include its content in this page, or apply a Quick Start Layout to add and configure an initial set of layout components.

File Name: catalog-custom-search.jsff

Directory: /ade/abhshsha_jfhiam2/iam/iam-product/consoles/IdentitySamples/CatalogSearchUI/public_html

Initial Page Layout and Content

( ) Blank Page

(•) Page Template: catalog-advanced-search-template

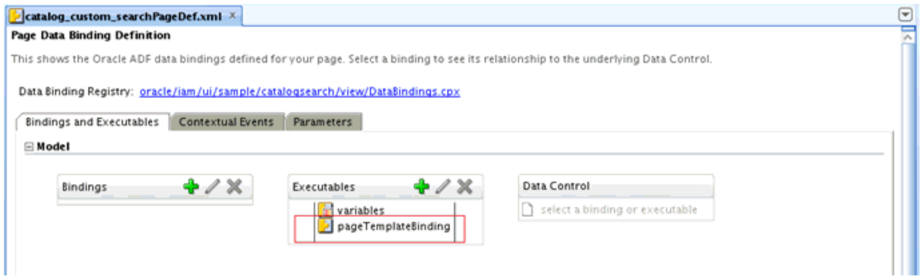
( ) Quick Start Layout

One Column (Stretched)

Page Implementation (UI components are not exposed in managed bean)

Make sure that the `pageTemplateBinding` is automatically added in the page definition of the custom page by Jdeveloper, as shown in Figure 19–9:

Figure 19–9 Page Bindings for JSF Page Fragments


catalog_custom_searchPageDef.xml

Page Data Binding Definition

This shows the Oracle ADF data bindings defined for your page. Select a binding to see its relationship to the underlying Data Control.

Data Binding Registry: oracle/iam/ui/sample/catalogsearch/view/DataBindings.cpx

3. Add the presentation logic for the custom form in the `search` facet, as shown:

```
<?xml version='1.0' encoding='UTF-8'?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
          xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
          xmlns:f="http://java.sun.com/jsf/core"
          xmlns:c="http://java.sun.com/jsp/jstl/core">
  <af:pageTemplate
viewId="/oracle/iam/ui/catalog/pages/catalog-advanced-search-template.jspx"
                   value="#{bindings.pageTemplateBinding}" id="pt1">
    <f:facet name="search">
      <!-- ######## CUSTOM SEARCH CONTENT BEGIN ###### -->
      <!-- ####### CUSTOM SEARCH CONTENT END########## -->
    </f:facet>
  </af:pageTemplate>
</jsp:root>
```

4. The search button on the custom form must have an action listener defined. In the action listener, construct the `SearchCriteria` object and invoke the following utility method:

```
oracle.iam.ui.catalog.view.CatalogAdvancedSearch.executeCatalogSearch(oracle.ia
m.platform.entitymgr.vo.SearchCriteria criteria)
public void searchActionListener(ActionEvent event){
    SearchCriteria criteria = null;
    //build your search criteria
```

```
        //and then call executeCatalogSearch
        CatalogAdvancedSearch.executeCatalogSearch(criteria);
    }
```

5. Deploy the taskflow as part of the `oracle.iam.ui.custom` shared library. For information about deploying the bounded taskflow, see "Deploying Custom Code to Oracle Identity Manager" on page 19-37.
6. Add permissions to the custom taskflow by using the Authorization Policy Manager (APM) UI to secure the taskflow.
7. Update the `Catalog Advanced Search Taskflow` system property to point to the custom taskflow. Specify the value in the format *TASKFLOW_DOCUMENT#TASKFLOW_ID*. See "System Properties in Oracle Identity Manager" in *Administering Oracle Identity Manager* for information about this system property.
8. Restart Oracle Identity Manager server.

19.7.9 Customizing Task Details Page for Approval Tasks

Before developing a custom task details taskflow, you must have the following software installed:

- Oracle Identity Manager 11*g* Release 2 (11.1.2.3.0)
- Oracle SOA Suite 11*g* (11.1.1.9.0)
- JDeveloper 11*g* (11.1.1.9.0) with Oracle SOA Composite Editor extension

Note: The same task details taskflow can be used for multiple human tasks as long as the human tasks have same set of outcomes and share the same payload structure. Therefore, different taskflows must be built for approvals, challenge, provide information, and manual fulfillment tasks.

To build a custom taskflow for a human task:

1. Create a ViewController project. For information about setting up the ViewController project, see "Setting Up the ViewController Project" on page 19-34.
2. Create the task details taskflow. To do so:
   a. Navigate to the following directory in the shiphome:

      IAM_HOME/server/workflows/composites/

   b. Unzip the composite:
      - For building details page for approvals and challenge tasks, unzip DefaultRequestApproval.zip.
      - For building details page for provide information tasks, unzip ProvideInformation.zip.
      - For building details page for manual fulfillment tasks, unzip DisconnectedProvisioning.zip.

   c. Go back to JDeveloper, right-click the ViewController project created in Step 1, and select **New**.
   d. Select **Web Tier**, **JSF**, and **ADF task flow** based on the human task.

e. In the file browser, navigate to the directory in which you unzipped the composite ZIP file. Select the appropriate human task file, as follows:

   – For building details page for approvals task, select **DefaultRequestApproval /ApprovalTask.task**.

   – For building details page for challenge task, select **DefaultRequestApproval/ChallengeTask.task**.

   – For building details page for provide information task, select **ProvideInformation/ApprovalTask.task**.

   – For building details page for manual fulfillment task, select **DisconnectedProvisioning /ManualProvisioningTask.task**.

f. In the Create Task flow dialog box, specify values for the following:

   File Name: For example, `request-approval-details-tf.xml`.

   Directory: Make sure that the taskflow is created under the `WEB-INF/oracle/iam/ui/custom/` directory. All taskflows under the `WEB-INF/oracle/iam/ui/custom/` directory are secured with view permission.

   Task Flow ID: For example, `request-approval-details-tf`.

g. Click **OK**.

3. Go to **Application Sources** under the ViewController project, and then delete hwtaskflow.xml.

4. Create the task details page. To do so:

   a. Open the taskflow created in Step 2. Switch to diagram mode.

   b. Rename taskdetails1_jspx view activity, for example, `request-approval-details`.

   c. Right-click the view activity, and select **Create Page**. Provide values for the following:

      File name: For example, `request-approval-details.jspx`

      Directory: Put the JSPX file under the `public_html/oracle/iam/ui/custom/` directory

      Initial Page layout and content: `Blank Page`

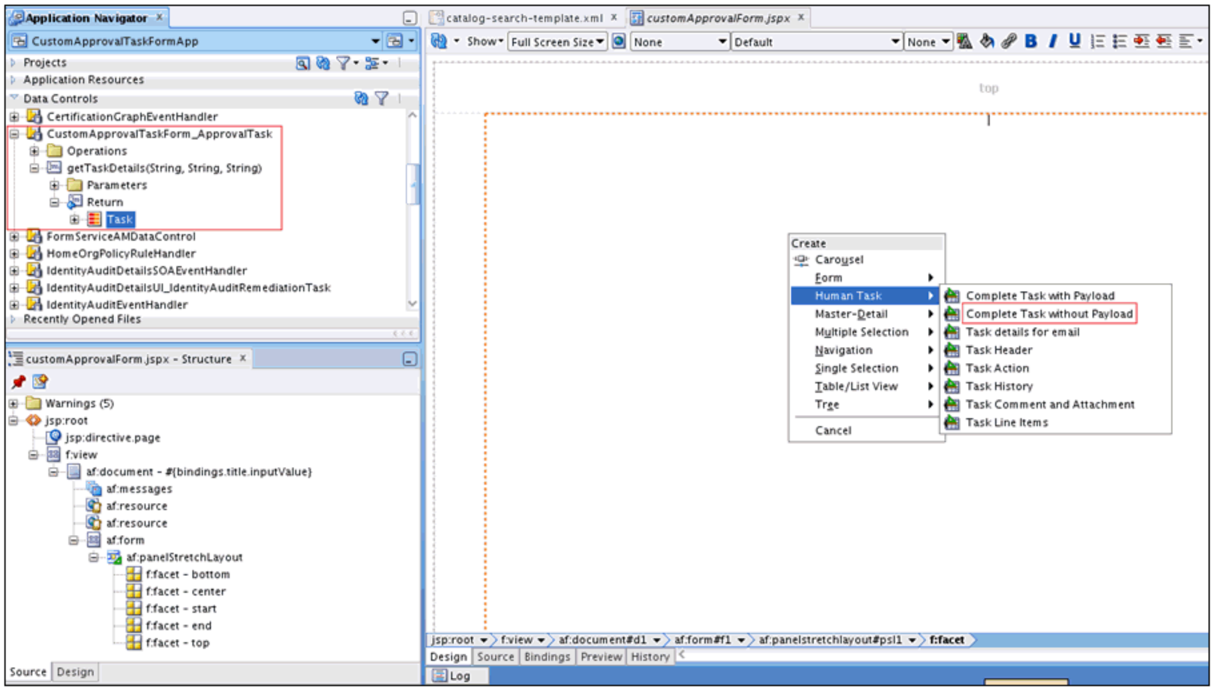
   d. Click **OK**.

5. Populate the page with task information. To do so:

   a. In the Data Controls palette, the data control with your project name is already created. Use this data control to render task-related information on the details page.

   b. To drop the Task object on the page, from the Create context menu, select **Human Task**, **Complete Task without Payload**, as shown in Figure 19–10.

Figure 19–10 Complete Task Without Payload



c. In the Edit Action dialog box, do not modify anything, and click **OK**.

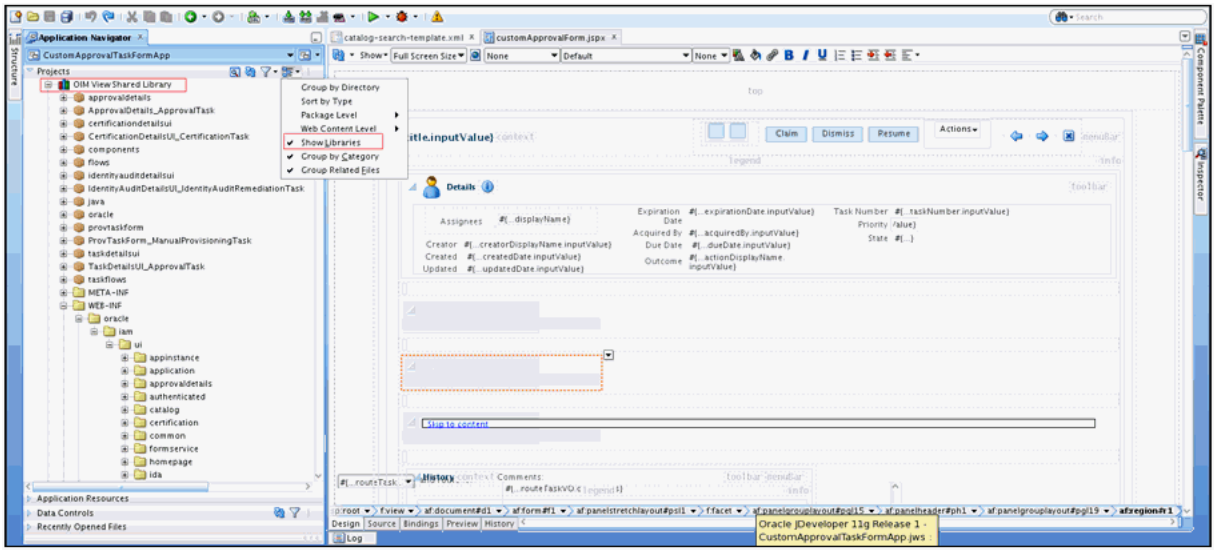
Note that task-related content, such as actions, basic information, history, comments, and attachments are added to the page.

Note: For manual fulfillment task, you can also choose the **Complete Task with Payload** option to show payload data. For approval, challenge, or provide information tasks, you can reuse Oracle Identity Manager taskflows mentioned in step 7 for showing payload-related information.

---

6. In the application navigator, enable the **Show Libraries** option, as shown in Figure 19–11. This allows you to drag and drop taskflows from OIM view shared library to your custom task details taskflow.

Figure 19–11 Enabling Show Libraries



7. Add request information to the Details page. The following taskflows from OIM View Shared library can be dropped as a region on the Details page to show request-related information.

   – WEB-INF/oracle/iam/ui/catalog/tfs/request-summary-information-tf.xml: This taskflow shows basic request information. This is not applicable for manual fulfillment task.

   – WEB-INF/oracle/iam/ui/catalog/tfs/request-summary-details-tf.xml: This taskflow shows target users, related requests, and dependent requests. This is not applicable for manual fulfillment task.

   – WEB-INF/oracle/iam/ui/catalog/tfs/catalog-tf.xml: This taskflow shows cart items and form data for each item in the request. It should be used for all request types except create, modify, or delete role. This taskflow can be used for approvals, challenge, provide information, and manual fulfillment tasks.

   – WEB-INF/oracle/iam/ui/role/tfs/create-role-train-tf.xml: This taskflow should be used instead of catalog-tf taskflow for create, modify, or delete role types of requests. It is not applicable for challenge, provide information, and manual fulfillment tasks.

8. Add a separate page for email notification (optional). By default, for sending email notification, if there is no separate page for email, then the same task details page developed in this section is sent in email notification. Sometimes, limited information needs to be sent in email notification. In such scenarios, separate page for email notification can be developed. The email page will also be part of the same task details taskflow. For more information on building custom page for email, refer to "Creating an Email Notification" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

9. Deploy the taskflow as part of the `oracle.iam.ui.custom` shared library. For information about deploying the bounded taskflow, see "Deploying Custom Code to Oracle Identity Manager".

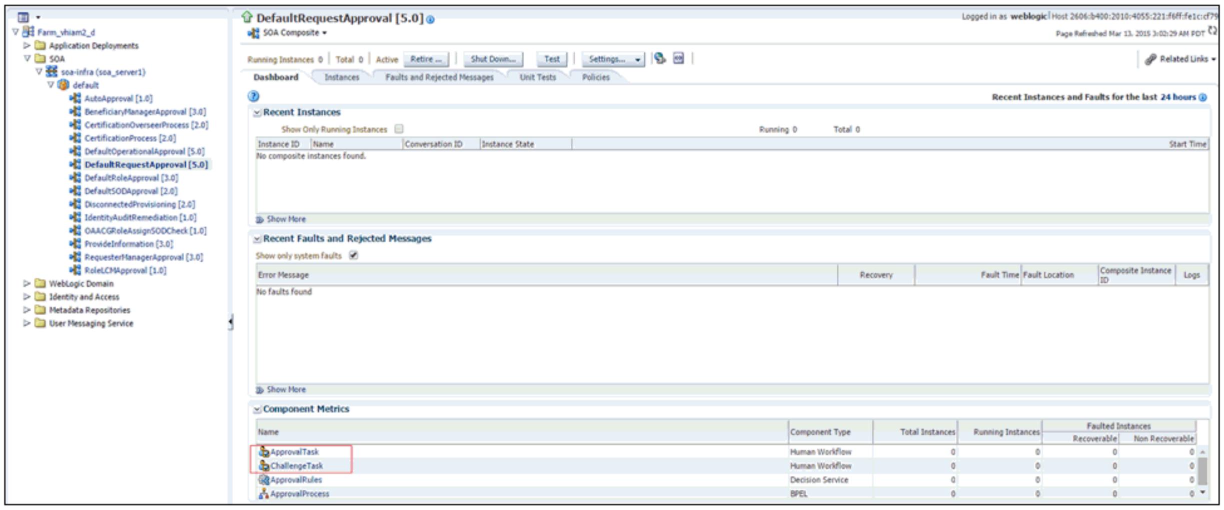
10. Configure the human task to use the custom taskflow. To do so:

    a. Login to Oracle Enterprise Manager as WebLogic user.

b. Navigate to **Farm_*IAM_DOMAIN***, **SOA**, **soa_infra (*SOA_SERVER*)**, **default**, ***COMPOSITE_NAME***. An example of composite name can be `DefaultRequestApproval[5.0]`.

c. Click **Component Metrics**, ***HUMAN_TASK***. Human task can be ApprovalTask or ChallengeTask or ManualProvisioningTask, as shown in Figure 19–12.

Figure 19–12 Human Tasks


Farm_vhiam2_d
Application Deployments
SOA
soa-infra (soa_server1)
default
AutoApproval [1.0]
BeneficiaryManagerApproval [3.0]
CertificationOverseerProcess [2.0]
CertificationProcess [2.0]
DefaultOperationalApproval [5.0]
DefaultRequestApproval [5.0]
DefaultRoleApproval [3.0]
DefaultSODApproval [2.0]
DisconnectedProvisioning [2.0]
IdentityAuditRemediation [1.0]
OAACGRoleAssignSODCheck [1.0]
ProvideInformation [3.0]
RequesterManagerApproval [3.0]
RoleLCMApproval [1.0]
WebLogic Domain
Identity and Access
Metadata Repositories
User Messaging Service
DefaultRequestApproval [5.0]
SOA Composite
Running Instances 0
Total 0
Active
Retire ...
Shut Down...
Test
Settings...
Related Links
Dashboard
Instances
Faults and Rejected Messages
Unit Tests
Policies
Recent Instances and Faults for the last 24 hours
Recent Instances
Show Only Running Instances
Running 0
Total 0
Show More
Recent Faults and Rejected Messages
Show only system faults
Show More
Component Metrics

Instance ID	Name	Conversation ID	Instance State	Start Time
No composite instances found.				

Error Message	Recovery	Fault Time	Fault Location	Composite Instance ID	Logs
No faults found					

Name	Component Type	Total Instances	Running Instances	Faulted Instances	
				Recoverable	Non Recoverable
ApprovalTask	Human Workflow	0	0	0	0
ChallengeTask	Human Workflow	0	0	0	0
ApprovalRules	Decision Service	0	0	0	0
ApprovalProcess	BPEL	0	0	0	0

d. Click the **Administration** tab.

e. Modify the URI in the existing entry to point to the custom taskflow, as follows:

Application Name: Worklist

Host Name: Hostname in OIMExternalFrontendURL

HTTP Port: HTTP Port in OIMExternalFrontEndURL if SSL is not configured

HTTPS Port: HTTPS Port in OIMExternalFrontEndURL if SSL is configured

URI: It is of the format /identity/faces/adf.task-flow?_id=TASKFLOW_ID&_document=TASKFLOW_DOCUMENT . For example, /identity/faces/adf.task-flow?_id=request-approval-details-tf&_document=WEB-INF/oracle/iam/ui/custom/request-approval-details-tf.xml

11. Repeat step 10 for all the human tasks for which you want to reuse this custom taskflow.

12. Restart all servers.

19.8 Configuring Additional Request Form

Users can enter additional information for a request that can be useful for the request approvers. The following topics are discussed in this section:

- Additional Request Information Concepts
- Understanding the Guidelines for Developing Custom Taskflow for Additional Request Information
- Configuring Custom Taskflow for Additional Request Information

- Validating Additional Request Information

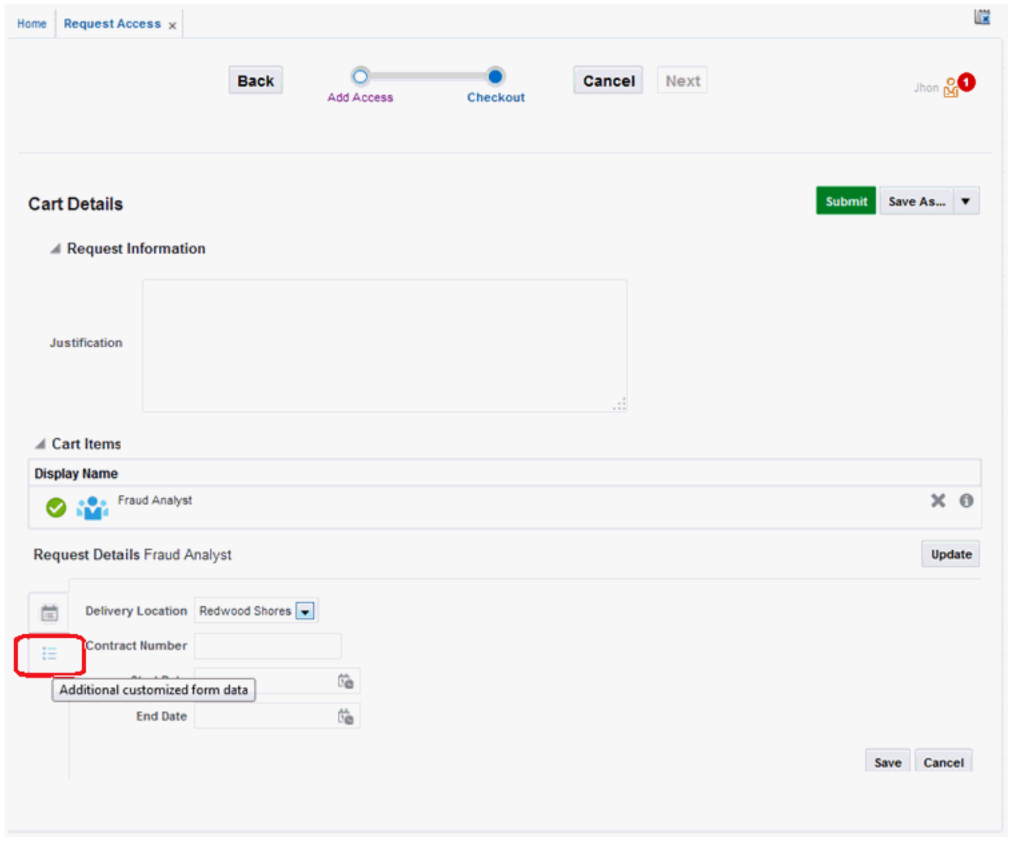
19.8.1 Additional Request Information Concepts

Additional information about the request cart item or about the request itself can be provided during request submission. It may be required for request approval decisions when default form information is not sufficient. The following features are supported with respect to additional information:

- Additional Information for the Request Cart Item
- Additional Information for the Request

19.8.1.1 Additional Information for the Request Cart Item

With custom taskflows developed, users can provide additional information about the cart item (role/account/entitlement) when raising a request. You can provide additional information form about the request cart item as shown in the following sample screenshot.



When the user clicks the additional info icon (marked in red), a form is displayed with custom fields. The user can enter appropriate information that is useful for the request approvers.

The request approver can view and/or update additional request information provided at the cart item level. The request approver can access the request from Identity Self Service on the Inbox page or the Pending Approvals page.

19.8.1.2 Additional Information for the Request

There are instances where additional information about the request is required, for example additional information about the modify user request. Unlike additional information at cart item level, there is no explicit placeholder in the cart submission UI for this type of additional information. However, after developing a custom taskflow, you can customize the cart details page to create a link to show additional information about the request.

19.8.2 Understanding the Guidelines for Developing Custom Taskflow for Additional Request Information

Developing a custom taskflow for additional request information involves the following:

- Implementing Custom Taskflow for Additional Request Information
- Saving and Retrieving Additional Information in Managed Bean Developed for the Project
- Understanding the AdditionalRequestInfo Interface
- Using RequestFormContext to Achieve the Required Customizations

19.8.2.1 Implementing Custom Taskflow for Additional Request Information

To implement a custom taskflow that can be used for providing and viewing additional request information:

1. Develop the custom taskflow as a bounded taskflow in ViewController projects. For information about setting up the ViewController project, see "Setting Up the ViewController Project" on page 19-34. Specify the default package as `oracle.iam.ui.custom`.
2. Create a taskflow to render additional form. Make sure that the taskflow is in the `WEB-INF/oracle/iam/ui/custom/` directory or its subdirectory, for example, `/WEB-INF/oracle/iam/ui/custom/sample/catalog/tfs/additional-role-info.xml#additional-role-info`.
3. Define taskflow input parameters as required.

   The following table lists predefined input parameters that are passed when the taskflow is launched:

Parameter Name	Type	Description
`additionalRequestInfo`	`oracle.iam.ui.common.model.catalog.requestdetails.AdditionalRequestInfo`	An instance of AdditionalRequestInfo interface. It is used to set and get values of additional request-level and cart item level request attributes. For more information about the interface methods, see "Understanding the AdditionalRequestInfo Interface" on page 19-76.

Parameter Name	Type	Description
`requestFormContext`	`oracle.iam.ui.platform.view.RequestFormContext`	An instance of RequestFormContext. RequestFormContext provides various context information related to the request and currently selected cart item. For example, you can leverage the information provided by RequestFormContext to decide whether the input components of the custom taskflow must be rendered in read-only or editable mode. For more information, see "Available EL Expressions in the RequestFormContext" on page 19-16.

4. Deploy the taskflow as part of the `oracle.iam.ui.custom` shared library. For information about deploying the bounded taskflow, see "Deploying Custom Code to Oracle Identity Manager" on page 19-37.

19.8.2.2 Saving and Retrieving Additional Information in Managed Bean Developed for the Project

The `additionalRequestInfo` pageFlowScope input parameter can be used to retrieve and store additional information about request or request cart item, as shown:

```
AdditionalRequestInfo additionalRequestInfo  =
(oracle.iam.ui.common.model.catalog.requestdetails.AdditionalRequestInfo)
AdfFacesContext.getCurrentInstance().getPageFlowScope().get("additionalRequestInfo
");
```

> **Note:** The `oracle.iam.ui.common.model.catalog.requestdetails.AdditionalRequestInfo` interface is described in "Understanding the AdditionalRequestInfo Interface" on page 19-76.

To set or get additional information about the request at the cart item level, use the following:

```
startDate = (Date additionalRequestInfo.getAttribute(cartItemId, START_DATE);
contractNumber = (String) additionalRequestInfo.getAttribute(cartItemId,
"CONTRACT_NUMBER" );
```

Similarly, to set the additional information at cart item level, use the following:

```
contractNumber = (String) additionalRequestInfo.setAttribute(cartItemId,
"CONTRACT_NUMBER" , "123");
```

Note: To retrieve the current cart ID item selected in this context, use the `requestFormContext` pageFlowScope input parameter, as shown:

```
RequestFormContext requestFormContext =
(oracle.iam.ui.platform.view.RequestFormContext)
AdfFacesContext.getCurrentInstance().getPageFlowScope().get("reques
tFormContext");
cartItemId = requestFormContext.getUniqueCartItemIdentifier();
```

To set or get additional information about the request, use the following:

```
temporaryLocation = (String) additionalRequestInfo.getAttribute("TEMPORARY_LOC");
additionalRequestInfo.setAttribute("TEMPORARY_LOC", "A-24");
```

19.8.2.3 Understanding the AdditionalRequestInfo Interface

AdditionalRequestInfo interface extends Serializable as shown in the following code. Therefore, an instance of AdditionalRequestInfo can be stored in pageFlowScope of the custom taskflow.

```
public interface AdditionalRequestInfo extends Serializable {
    public void setAttribute(String name, Serializable value);
    public void setAttribute(String cartItemId, String name, Serializable value);
    public Serializable getAttribute(String name);
    public Serializable getAttribute(String cartItemId, String name);
}
```

To get or set the individual cart item additional information, use the following:

```
public void setAttribute(String cartItemId, String name, Serializable value);
public Serializable getAttribute(String cartItemId, String name);
```

The `cartItemId` parameter must be specified. You can use the `getUniqueCartItemIdentifier` EL expression in the RequestFormContext to access the selected `cartItemId`. For more information, see "Available EL Expressions in the RequestFormContext" on page 19-16.

To get or set the request level additional information, use the following:

```
public void setAttribute(String name, Serializable value);
public Serializable getAttribute(String name);
```

19.8.2.4 Using RequestFormContext to Achieve the Required Customizations

To distinguish between the various flows in which the taskflow is being launched (cart submission, request details, and approval details flows), you can rely on the `requestFormContext` parameter that is passed to the custom taskflow.

For instance, if you want to show the components of additional information taskflow to be in read-only mode for request summary, then use the following:

```
if  (requestFormContext.getActionType() == requestFormContext.ActionType.APPROVAL)
{
    isFormUpdateable = Boolean.TRUE;
}
```

19.8.3 Configuring Custom Taskflow for Additional Request Information

Configuring custom taskflow for additional request information is described in the following sections:

- Configuring Custom Taskflow for the Cart Item Level
- Configuring Additional Request Information at Request Level

19.8.3.1 Configuring Custom Taskflow for the Cart Item Level

The cart submission UI has explicit placeholders where the additional cart item information can be displayed.

You can enable users to provide additional information specific to a cart item, such as application instance, role, or entitlement. However, you are allowed to configure only one additional information taskflow per each entity type: role, entitlement, or application instance. This configuration is done with the help of the following system properties:

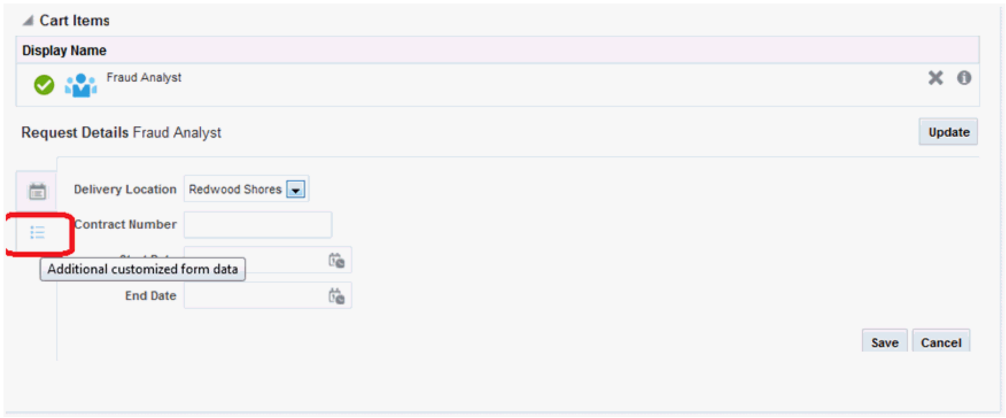
- **Catalog Additional Application Details Task Flow:** Set the value to the custom additional information taskflow that is applicable to application instance entity type.
- **Catalog Additional Entitlement Details Task Flow:** Set the value to the custom additional information taskflow that is applicable to entitlement entity type.
- **Catalog Additional Role Details Task Flow:** Set the value to the custom additional information taskflow that is applicable to role entity type.

> **Note:** For more information on setting system properties, see "Managing System Properties" in *Administering Oracle Identity Manager*.

For example, to launch a specific additional information taskflow for the role entity type, the value of the `Catalog Additional Role Details Task Flow` system property can be set as follows:

```
/WEB-INF/oracle/iam/ui/custom/sample/catalog/tfs/additional-role-info.xml#
additional-role-info
```

The Cart Items table is displayed in the Cart Checkout, Request Summary, and Request Approval pages whenever the UI operation involves cart items. Clicking the cart item's additional info icon, the custom taskflow configured for cart entity type (such as role, application instance, entitlement) is displayed, as shown in the following sample screenshot:



Note: It is not recommended to use the approach applicable to Oracle Identity Manager 11*g* Release 2 (11.1.2.2.0) for configuring additional information at cart item level.

19.8.3.2 Configuring Additional Request Information at Request Level

There are some scenarios when additional information is required at the request level, for example while submitting user modification request, where requester can provide additional information about user, and approver can review those. This feature is not available in the Identity Self Service by default, and must be implemented by developing custom taskflow, creating a sandbox, and UI customization, and establishing a link between the custom taskflow and the UI.

You can enable users to provide additional information specific to a request, such as Modify User request. To do so, you can add a link or a button to the catalog checkout page by customizing the UI. The newly added link or button can be made visible to Cart Checkout, Request Summary, and Request Approval pages. On clicking the link or button, you can invoke the popup to display the additional information about request.

A set of attributes defined as descendants of the component determine which custom taskflow is to be launched and how the popup window is displayed. The following table lists the predefined attributes:

Name	Type	Required	Description
taskFlowId	java.lang.String	Yes	ID of the custom taskflow to be launched by the component.
dialogTitle	java.lang.String	Optional	Title of the popup window in which the custom taskflow is launched. If no value is specified, then the title is blank.
dialogTitleIcon	java.lang.String	Optional	Path to the icon that is displayed in the popup window in which the custom taskflow is launched. If no value is specified, then the default icon is used.

The following is a sample code snippet for the Command Link added through UI customization to invoke Additional Request Information popup:

Note: For information about adding a Command Link through UI customization, see "Adding a Link or Button" on page 19-20.

```
<af:commandLink xmlns:af="http://xmlns.oracle.com/adf/faces/rich" id="e8065932664"
text="Additional Cart Information"
actionListener="#{catalogRequestBean.launchAdditionalRequestInfoTaskFlow}"
rendered="#{backingBeanScope.additionalInfoHelperBean.renderAdditionalDetailsForRe
quest}">
         <af:clientAttribute xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
name="taskFlowId"
value="#{backingBeanScope.additionalInfoHelperBean.additionalInfoTaskFlowIdForRequ
est}"/>
         <af:clientAttribute xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
name="dialogTitleIcon" value="/images/request_ena.png"/>
         <af:clientAttribute xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
name="headerText" value="Additional Cart Information"/>
         <af:clientAttribute xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
name="dialogTitle"
value="#{backingBeanScope.additionalInfoHelperBean.popupTitle}"/>
</af:commandLink>
```

In this sample code snippet, a custom Additional Cart Information link is added to cart details page. Here:

- The actionlistener for the command link must be set to `catalogRequestBean.launchAdditionalRequestInfoTaskFlow` as follows:

  ```
  actionListener="#{catalogRequestBean.launchAdditionalRequestInfoTaskFlow}"
  ```

- The custom link can be selectively displayed for the rendered property by using the EL expression:

  ```
  #{backingBeanScope.additionalInfoHelperBean.renderAdditionalDetailsForRequest}
  ```

  To use this EL expression, develop a managed JAVA bean, such as `additionalRequestInfoHelperBean`, with the `isRenderAdditionalDetailsForRequest` method. This method returns a boolean value (true or false) based on whether the link has to be displayed or not. For more information about the managed bean, see "Developing Managed Beans and Task Flows" on page 19-34.

- Launch different taskflows using the EL expression, `#{backingBeanScope.additionalInfoHelperBean.additionalInfoTaskFlowIdForRequest}`, specified as the value of the `taskFlowId` attribute. This is a mandatory attribute.

  To use this EL expression, include the `getAdditionalInfoTaskFlowIdForRequest` method in the managed bean named `additionalRequestInfoHelperBean`. This method returns a String value representing the ID of the custom taskflow deployed as part of `oracle.iam.ui.custom-dev-starter-pack.war`, for example, `/WEB-INF/oracle/iam/ui/custom/catalog/tfs/additional-cart-info-tf.xml#additional-cart-info-tf`.

- Set different popup window titles using the EL expression, `#{backingBeanScope.additionalInfoHelperBean.popupTitle}`, specified as the value of the `dialogTitle` attribute.

To use this EL expression, include the `getPopupTitle` method in the managed bean named `additionalRequestInfoHelperBean`. This method returns the desired String value to be displayed as the popup window title.

19.8.4 Validating Additional Request Information

If the additional request information has any mandatory attribute values to be submitted, then you can validate the submission by using a `RequestDataValidator` plug-in, which can validate RequestData.

The following is a sample configuration for the Assign Roles operation in the `plugin.xml` file while registering the validator plug-in. This configuration ensures that `mycompany.iam.plugin.validation.AssignRolesDataValidator` is invoked for all Assign Roles operations.

```
<plugins pluginpoint="oracle.iam.request.plugins.RequestDataValidator">
    <plugin pluginclass="mycompany.iam.plugin.validation.AssignRolesDataValidator"
version="1.0" name="AssignRolesDataValidator">
           <metadata name="DataValidator">
                  <value>AssignRolesDataset</value>
           </metadata>
    </plugin>
</plugins>
```

Similarly, for the Assign Entitlements operation, `plugin.xml` can be configured as follows:

```
<plugins  pluginpoint="oracle.iam.request.plugins.RequestDataValidator">
    <plugin
pluginclass="mycompany.iam.plugin.validation.AssignEntitlementsDataValidator"
version="1.0" name="AssignEntitlementsDataValidator">
           <metadata name="DataValidator">
                  <value> EBSForm.UD_EBS_RESP</value>
           </metadata>
    </plugin>
</plugins>
```

19.9 Migrating UI Customizations

Migrating UI customizations from one Oracle Identity Manager environment to another environment or test to production (T2P) is described with the help of the following scenarios:

Scenario I: Incremental T2P

During the development cycle, you want to incrementally build configuration and keep moving the configuration from one Oracle Identity Manager setup to another. To do this, you use the Deployment Manager, as described in "Migrating Incrementally Using the Deployment Manager" in *Administering Oracle Identity Manager*. But exporting and importing data using the Deployment Manager does not include the UI customization. For this reason, Oracle Identity Manager provides sandboxes, using which you can create customizations bound by sandboxes, test them, and eventually export/import them on an incremental basis.

However, incremental migration of customizations has a problem. You have to keep your sandboxes exported in advance, and then only publish the changes. But if you have already published the changes, then you cannot export. This is a known issue.

Scenario II: Fusion Middleware Framework-Based Full T2P

After completion of the development and testing cycles, you want to setup the production environment on the first day of the real deployment. You want to move all configurations and customizations from the test to the production environment.

Full T2P of Oracle Identity Manager via Fusion Middleware framework-based utility also supports the movement of UI customizations. Fusion Middleware Framework-Based Full T2P is performed by using Fusion Middleware T2P utilities, such as copyBinary, pasteBinary, copyconfig, extractMoveplan, and pasteconfig, and does not use the Deployment Manager. See "Moving from a Test to a Production Environment" in the *Oracle Fusion Middleware Administrator's Guide* for detailed information about Full T2P.

If in the T2P, the list of published sandboxes are not showing up, then it is not an issue because you are expected to track published/unpublished changes in your test environment (T2P source), not in the destination or production environment.

Any unpublished sandboxes in the source or test environment means:

- You want to move the unpublished sandboxes later as incremental work after increasing the scope of those sandboxes, but currently those have not been included in the production environment.
- The unpublished sandboxes have not been tested, and you do not want to include those in the production environment.

19.10 UI Customization Best Practices

This section describes the following UI customization best practices and guidelines:

Create sandboxes with detailed description

When creating a sandbox, create it with a detailed description and list all the entities for which you are creating the sandbox. For example, if you are creating an application instance, note that this sandbox is created for application instance creation. When the application instance is created, publish the sandbox, and then go to Identity Self Service to create another sandbox to perform the UI customization. This is to avoid issues when two or more users create different sandboxes to create the same entity (application instance in this example) and try to publish it at different times.

Create a backup of MDS before publishing a sandbox

Before publishing a sandbox, create a backup of MDS.

Migrate all sandboxes to the target environment and publish in the same order

All the sandboxes that have been published in the first environment (or source environment) must be migrated and published in the second environment (or target environment), and the sandboxes must be published in the same order. If this guideline is not followed, then some of the customizations will be missing in the target environment resulting in ADF errors or missing attribute display labels.

Migrating sandboxes from multiple source environments into a single target environment is not supported. See sections "Handling Concurrency Conflicts" on page 19-3 and "Troubleshooting Concurrency Issues" on page 19-4 for detailed information about various scenarios.

Export the sandbox before publishing

If you are planning to migrate customizations from one environment to another, then all the sandboxes must be exported before publishing. It is not possible to export a sandbox that has already been published.

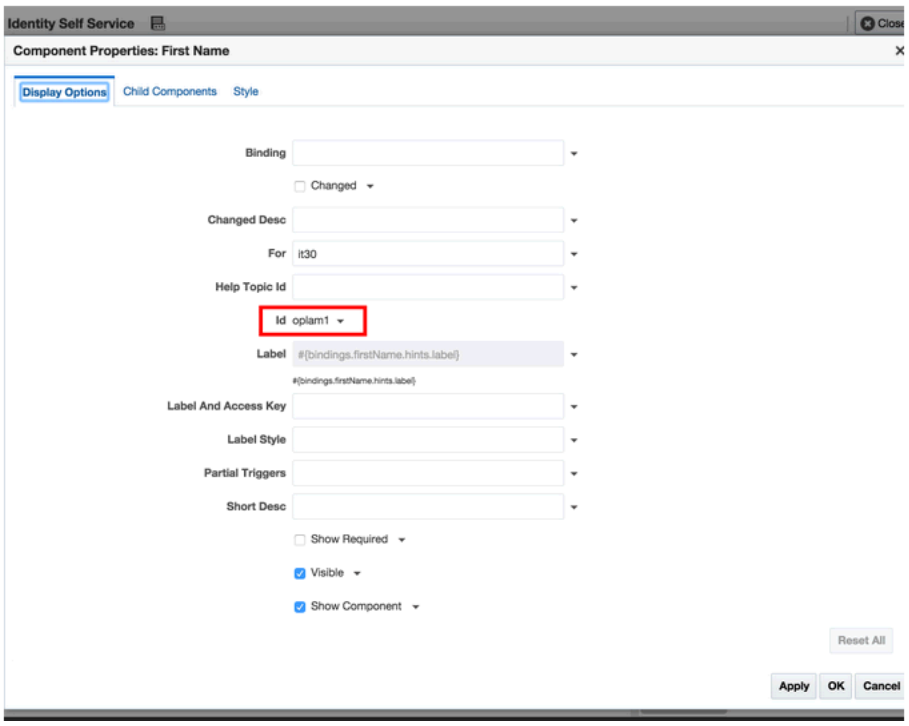
Test the sandbox before publishing

The main purpose of using sandboxes is to be able to experiment with customizations. Therefore, publish a sandbox only after thorough testing of related use cases. After the sandbox is published, it cannot be unpublished easily. This also applies to migration of sandboxes to another environment, where the sandbox must be published only after sanity testing.

Do not change default component IDs

It is possible to view and edit component IDs by using WebCenter Composer, as shown in Figure 19–13.

Figure 19–13 Component ID



Component Properties: First Name

Binding	
☐ Changed	
Changed Desc	
For	it30
Help Topic Id	
Id	oplam1
Label	#{bindings.firstName.hints.label}
	#{bindings.firstName.hints.label}
Label And Access Key	
Label Style	
Partial Triggers	
Short Desc	
☐ Show Required	
✓ Visible	
✓ Show Component	

Note that changing IDs of default components on system-defined pages must be avoided. Component ID must be treated as read-only. It can be used, for example, in the `Partial Triggers` property of another component if the component is supposed to be re-rendered based on changes or when click to the component is being referenced.

Use discretion when deleting components from a page

Exercise caution when deleting system-defined components from pages, especially when the component binding property is set. These components can be referenced from the managed beans shipped with Oracle Identity Manager, and removing the component from the page will result in bindings not being set. This can lead to errors, such as `NullPointerException`. In such instances, it is preferable to hide the component from the page by setting the `Visible` property to `false`.

Note that marking a component as not being rendered (deselected **Show Component** option) has the same effect on component bindings, which means it will not be set.

Note that direct changes to default EOs/VOs are not supported

Making any direct changes, such as exporting the EO/VO XML from MDS and modifying it, to system-defined EOs/VOs, such as UserVO, OrganizationVO, and RoleDetailsVO, is not supported. The Form Designer, which accessed via User, Organization, Role, or Catalog system entities links in the Identity System Administration, is the only supported way of adding new attributes or modifying existing ones.

Specify name space for JSFF tags

If you are manually adding JSFF components in an exported sandbox, then specify name space for each of the JSFF tags. The following code snippet shows an example customization document where new `af:outputText` is being added:

```
<?xml version='1.0' encoding='UTF-8'?>
<mds:customization version="11.1.1.66.49" xmlns:mds="http://xmlns.oracle.com/mds"
motype_local_name="root" motype_nsuri="http://java.sun.com/JSP/Page">
   <mds:insert parent="sdh1" position="first">
      <af:outputText value="Some text" id="e7869964958"/>
   </mds:insert>
</mds:customization>
```

Note that there is no name space specified for `af:outputText`. Such code snippet will cause `java.io.IOException: Stream closed` when user opens the page. No exception is thrown during sandbox import or when activating the sandbox. The exception is thrown only when the page is being accessed.

The following code snippet shows the corrected `af:outputText` with `xmlns:af="http://xmlns.oracle.com/adf/faces/rich"` name space specified:

```
<?xml version='1.0' encoding='UTF-8'?>
<mds:customization version="11.1.1.66.49" xmlns:mds="http://xmlns.oracle.com/mds"
motype_local_name="root" motype_nsuri="http://java.sun.com/JSP/Page">
   <mds:insert parent="sdh1" position="first">
      <af:outputText xmlns:af="http://xmlns.oracle.com/adf/faces/rich" value="Some
text" id="e7869964958"/>
   </mds:insert>
</mds:customization>
```

Note that customizations are only allowed in site/site layer

If you are manually editing an MDS customization document, for example, in an exported sandbox, then you are only allowed to edit documents in the `site/site` customization layer. All other customization layers, such as `site/oim`, `edition/xe`, and `edition/ee`, must not be used.

You can find out from the document path which layer the document resides in. For example, the `oracle/iam/ui/myinformation/pages/mdssys/cust/site/site/my-info.jsff.xml` document resides in the `site/site` layer. The layer name is specified after `mdssys/cust`.

All the customizations made by using WebCenter Composer are stored in the `site/site` layer. Home page personalizations and other personalizations are stored in `user/`*`USER_NAME`* layer. While personalization documents can be manually edited, they are only applicable to the *`USER_NAME`* user.

Note that each application instance or entitlement form has three page fragments (JSFF)

Every application instance or entitlement form has the following page fragments (JSFF):

- **create:** Displayed when requesting for application instance or entitlement
- **modify:** Displayed when modifying existing application instance or entitlement; also used when viewing application instance or entitlement details on application instance or entitlement pages in user profile and My Access sections of Identity Self Service
- **bulk:** Displayed for bulk requests

Whenever you are customizing the application instance or entitlement form, you typically want to customize all three page fragments (JSFF), or at least create and modify page fragment. Create page fragment can be customized when requesting for application instance or entitlement, and modify page fragment can be customized when modifying existing application instance or entitlement. All three page fragments are regenerated when you click the **Regenerate View** button in the Form Designer.

Use Discretion when using the Searchable Picklist option

When creating lookup UDFs you have an option to select the **Searchable Picklist** option. This option must be checked only if you are planning to use `Input List of Values` component when adding the UDF to the page, which you must decide at the time of creating the UDF because the value of Searchable Picklist cannot be changed later. If you decide to select the **Searchable Picklist** option, then there is an additional VO attribute created. The attribute is suffixed with `__c_Id__c` and is for internal use only. When adding the UDF to the page, select and add the regular attribute, not the one with `__c_Id__c` suffix.

If you did not select the **Searchable Picklist** option, then the correct component to choose when adding the UDF to the page is `Select One Choice` and not `Input List of Values`.

Sign-out after adding/updating UDF

You must sign-out from Identity Self-Service or Identity System Administration after adding new or updating existing UDF. This is to avoid known caching issue in ADF layer wherein older version of the VO is being cached and new changes are not being picked up.

If you forget to sign-out and go directly to the page where the VO is being used, you will see an error similar to `JBO-25058: Definition MyUDF__c of type Attribute is not found in UserVO`, or you will not be able to select the UDF in WebCenter Composer catalog while adding the UDF to the page.

Verify the UDF after adding it to the page

If you add a UDF to the page and the UDF is not working, for example the input component is shown as read-only although it should be editable or the provided UDF value is not being properly saved, then you can verify the following:

- Make sure that you use the right Data Component and VO when adding the UDF to the page, as described in "Entities and Corresponding Data Components and View Objects" in *Administering Oracle Identity Manager*.
- Check if the UDF has been properly created. For the UDF that is created, there must be a column created in the corresponding database table, for example in the

USR table for User UDF, and corresponding dataset must be updated, for example User.xml for User UDF.

If any of these have not been or is missing, then the UDF is not properly created and must be created again.

If you forget to set `autoSubmit=true` and set `valueChangeListener` on the UDF component, as described in "Adding a Custom Attribute" in *Administering Oracle Identity Manager*, then the UDF works but the **Save**/**Cancel** or **Apply**/**Revert** buttons are not enabled.

Map UDF with correct LDAP attribute

While creating a UDF in Oracle Identity Manager deployment in LDAP mode, map the UDF with the correct LDAP attribute.

Deploy custom managed beans as part of the oracle.iam.ui.custom-dev-starter-pack.war shared library

When introducing custom managed beans, the beans must be deployed as part of the `oracle.iam.ui.custom-dev-starter-pack.war` shared library. Recommended scope of the custom managed beans is either `pageFlowScope` or `backingBeanScope`. Avoid defining custom beans in `sessionScope` and consider using `pageFlowScope` instead. The `pageFlowScope` bean class must be serializable. Implement `java.io.Serializable` interface and all the class fields must be serializable as well. Note that component bindings, such as `RichOutpuText`, are not serializable, and therefore, must be defined in `backingBeanScope` bean. If you are implementing a custom task flow, then the recommended practice is to have two managed beans:

- `pageFlowScope` bean, which holds the state of the taskflow (if any).
- `backingBeanScope` bean for component bindings `actionListeners` (or listeners in general). The `backingBeanScope` bean can access `pageFlowScope` bean and its properties via an EL, not vice versa.

Consider replacing the entire taskflow

If you want to significantly change the default look and feel of the page, then it might be beneficial to completely re-implement the entire taskflow. This way it is guaranteed that your custom taskflow will work even after upgrading Oracle Identity Manager. This may not be true if you significantly customize one of the predefined pages. Although Oracle Identity Manager UI customizations are upgrade-safe, there are exceptions to this when customizations are broken or lost post-upgrade. This is because it is sometimes not possible to retain customizations if the flow changes completely.

If you decide to re-implement the entire taskflow, then you can add a new Home page tile on one of the Home pages to launch the new taskflow and hide the original tile by launching a default taskflow. See "Adding a Tile to the Home Page" on page 19-29 for more information.

Do not update Oracle Identity Manager WAR/EAR files

You must not updated the following WAR/EAR files:

- `oracle.iam.console.identity.self-service.ear`
- `oracle.iam.console.identity.sysadmin.ear`
- `oracle.iam.ui.view.war`
- `oracle.iam.ui.oia-view.war`

- `oracle.iam.ui.model.ear`

Any changes to one of these WAR/EAR files are lost during upgrade.

The only WAR file that you can update is `oracle.iam.ui.custom-dev-starter-pack.war`. In fact, this WAR file is intended for use in customizations, such as in custom managed beans, resource bundles, and taskflows. Changes to `oracle.iam.ui.custom-dev-starter-pack.war` are retained during upgrade.

Consider conditionally showing certain home page tiles

By default, all the home page tiles are displayed. However, you might want to consider hiding some of the tiles from certain users to prevent them from accessing the pages. For example, you might want to hide the Provisioning Tasks tile for end users. See "Showing Tiles Conditionally" on page 19-34 for details.

Do not invoke Platform APIs from custom managed bean

Invoking Platform APIs directly by using Oracle Identity Manager data source in custom managed bean is not supported. Only Public APIs that are exposed through OIMClient can be invoked.

Use recommended value of Display Width while creating lookup UDFs

While creating lookup type UDF, the recommended value of the Display Width field is `40`.

Part VII

Interfaces to Integrate With Other Applications

This part describes the APIS and Web services that Oracle Identity Manager supports.

It contains the following chapters:

20 Using APIs

Oracle provides a network-aware, Java-based application programming interface (API) that exposes Services, called Utility in earlier releases, available in Oracle Identity Manager. This API is based on Plain Old Java Objects (POJO) and takes care of all the plumbing required to interact with Oracle Identity Manager. This API can be used for building clients for Oracle Identity Manager and for integrating third-party products with the Oracle Identity Manager platform.

Note: In this release, Oracle Identity Manager does not support the following:

- Legacy APIs
- Signature-based login

This chapter contains these sections:

- Accessing Oracle Identity Manager Services
- Oracle Identity Manager Services
- Commonly Used Services
- Developing Clients for Oracle Identity Manager
- Working With Legacy Oracle Identity Manager APIs
- Code Samples

20.1 Accessing Oracle Identity Manager Services

The entry point to Oracle Identity Manager Services is through oracle.iam.platform.OIMClient class. Thor.API.tcUtilityFactory used in earlier releases is also supported. Oracle recommends using the oracle.iam.platform.OIMClient for developing clients to integrate with Oracle Identity Manager.

This section describes the following topics:

- Using OIMClient
- Using OIMClient and tcUtilityFactory in Integrated Deployments

20.1.1 Using OIMClient

OIMClient is the entry point for accessing the services available in Oracle Identity Manager. You use the following sequence of steps when using OIMClient:

1. Create an instance of OIMClient with the environment information required to connect to Oracle Identity Manager application, as shown:

   ```
   Hashtable env = new Hashtable();

   env.put(OIMClient.JAVA_NAMING_FACTORY_INITIAL,
   "weblogic.jndi.WLInitialContextFactory");
   env.put(OIMClient.JAVA_NAMING_PROVIDER_URL, t3://OIM_HOSTNAME:OIM_PORT);
   OIMClient oimClient = new OIMClient(env);
   ```

   Here, replace *OIM_HOSTNAME* with the host name on which Oracle Identity Manager is deployed and *OIM_PORT* with the port number.

2. Login to the Oracle Identity Manager with the appropriate credentials, as shown:

   ```
   oimClient.login(OIM_USERNAME, OIM_PASSWORD);
   ```

3. Lookup a service, as shown:

   ```
   UserManager usermgr = oimClient.getService(UserManager.class);
   OR
   tcLookupOperationsIntf lookupIntf =
   oimClient.getService(tcLookupOperationsIntf.class);
   ```

4. Call method on a service, as shown:

   ```
   HashMap userAttributes = new HashMap();
   ……………..
   UserManagerResult result = userMgr.create(new User(null, userAttributes));
   ```

20.1.2 Using OIMClient and tcUtilityFactory in Integrated Deployments

In Oracle Identity Manager deployment that is integrated with Access Manager (OAM), OIMSignatureAuthenticator is not configured in the Oracle Identity Manager domain's security realm. Therefore, all the custom or partner applications that you want to integrate with Oracle Identity Manager must not use signature-based login to Oracle Identity Manager. Instead, you must follow any one of the following approaches:

- **Oracle Platform Security Services (OPSS) Framework:** If the partner or client application is a J2EE application based on Fusion Middleware stack, then it can use the following from OPSS framework:

  Note: See "Introduction to Oracle Platform Security Services" in the *Oracle Fusion Middleware Application Security Guide* for information about OPSS and its main features

  ---

  – **OPSS credential store:** This allows credentials to be managed (store, retrieve, modify) in a secure manner. You can store the password in the OPSS credential store, and retrieve it while performing a Oracle Identity Manager client login by using user ID and password. See "Managing the Credential Store" in the *Oracle Fusion Middleware Application Security Guide* for more information about OPSS credential store.

– **OPSS SubjectSecurity API:** If a partner application wants to invoke Oracle Identity Manager EJB/Service APIs with a higher privilege, such as system administrator user, then OPSS SubjectSecurity API can be used. The following sample partner application code tries to invoke Oracle Identity Manager API with higher privilege:

  See Also: *Oracle Fusion Middleware Java API Reference for Oracle Platform Security Services* for detailed information about the SubjectSecurity API

```
//Get ActionExecutor for OIM System administrator, xelsysadm
ActionExecutor actionExecutor =
SubjectSecurity.getInstance().getActionExecutor("xelsysadm");
actionExecutor.execute(new PrivilegedAction<Object>() {
  public Object run() {
       //OIM EJB method invocation goes here….
                   Hashtable env = new Hashtable();
      //serverURL - OIM server's RMI URL
                   // ctxFactory - WLS/WAS context factory class
         env.put(OIMClient.JAVA_NAMING_PROVIDER_URL, serverURL);
         env.put(OIMClient.JAVA_NAMING_FACTORY_INITIAL,ctxFactory);
         OIMClient client = new OIMClient(env);
  //Invoking EJB service method as "xelsysadm"
         RequestService reqSrvc = client.getService(RequestService.class);
         reqSrvc.getBasicRequestData("1");//1 is the request ID.
              }
});
```

- If using the OPSS framework is not possible for some reason, then it is recommended to invoke the OIMClient API with user ID and password. However, it is up to the client or partner to store and manage the Oracle Identity Manager user's password in a secure manner.

20.2 Oracle Identity Manager Services

The Oracle Identity Manager API provides access to services available in Oracle Identity Manager. Because the APIs in Oracle Identity Manager 11*g* Release 1(11.1.1) onwards and the legacy APIs use different conventions, this section discusses them separately in the following topics:

- Services in Oracle Identity Manager 11g
- Legacy Services or Utilities

20.2.1 Services in Oracle Identity Manager 11*g*

Services in Oracle Identity Manager 11*g* onwards are based on the following conventions:

- **Package Names:** Services are in packages whose names end with "api", for example:

```
oracle.iam.request.api
oracle.iam.identity.usermgmt.api
```

- **Service Interface Names:** Services introduced in 11*g* typically use the naming convention of "*Service", for example:

```
oracle.iam.request.api.RequestService
oracle.iam.selfservice.self.selfmgmt.api.AuthenticatedSelfService
```

Some Identity Administration APIs use the "*Manager" naming convention for their APIs, for example:

```
oracle.iam.identity.usermgmt.api.UserManager
```

Some new services introduced in Oracle Identity Manager 11*g* Release 2 (11.1.2.3.0) are:

```
oracle.iam.api.OIMService
oracle.iam.platform.authopss.api.AuthorizationService
oracle.iam.provisioning.api.ProvisioningService
oracle.iam.provisioning.api.ApplicationInstanceService
```

20.2.2 Legacy Services or Utilities

Legacy services, also called utilities, follow the following naming conventions

- **Package Names:** All legacy APIs are in Thor.API.Operations package.
- **Service Interface Names:** Service names are of the form "*Intf", for example, Thor.API.Operations.tcImportOperationsIntf.

See Also: *Oracle Fusion Middleware Java API Reference for Oracle Identity Manager* for a full list of services available in Oracle Identity Manager. You can use the naming conventions above to find the APIs.

20.3 Commonly Used Services

Table 20–1 lists some commonly used services in Oracle Identity Manager.

Table 20–1 Commonly Used Services

Service Name	Description
UserManager	Provides operations for user management, such as create, search, modify, and delete users
RequestService	Provides operations to submit, withdraw, close, and search requests.
RoleManager	Provides operations for role management such as create, search, modify, and delete roles. In addition, this service provides operations for management of role members and relationships between roles.
OrganizationManager	Provides operations for organization management such as create, search, modify, delete, enable, and disable organizations.
oracle.iam.api.OIMService	Provides method to perform an operation in Oracle Identity Manager. You can pass an intent while calling API of this service. Intent here can be request or direct.

20.3.1 Mapping Between Legacy and New Services

In Oracle Identity Manager, some of the legacy APIs have been rewritten by using new architecture and the corresponding utility services or interface classes have been changed. Table 20–2 provides a high-level correspondence between the legacy and new interfaces.

Table 20–2 Mapping Between Legacy and New Services

Legacy Service	New Service
Thor.API.Operations.tcUserOperationsIntf	oracle.iam.identity.usermgmt.api.UserManager
Thor.API.Operations.tcGroupOperationsIntf **Note:** The Group Manager APIs related all delegated admin APIs for adding and removing admins have been deprecated.	oracle.iam.identity.rolemgmt.api.RoleManager
Thor.API.Operations.tcOrganizationOperationsIntf	oracle.iam.identity.orgmgmt.api.OrganizationManager
Thor.API.Operations.tcRequestOperationsIntf	oracle.iam.request.api.RequestService
Thor.API.Operations.tcSchedulerOperationsIntf	oracle.iam.scheduler.api.SchedulerService
Thor.API.Operations.tcEmailOperationsIntf	oracle.iam.notification.api.NotificationService

20.4 Developing Clients for Oracle Identity Manager

This section includes the following topics:

- Prerequisites for Developing Clients
- Setup and Configuration

20.4.1 Prerequisites for Developing Clients

The following prerequisites must be met for developing clients for Oracle Identity Manager:

- Java Development Kit (JDK) 1.6 installed and set in the path
- ANT 1.7 installed and set in the path

20.4.2 Setup and Configuration

Oracle Identity Manager package contains a ZIP file that contains the required libraries and configuration files for developing clients.

To run an application client for Oracle Identity Manager:

1. Copy *OIM_ORACLE_HOME*/server/client/oimclient.zip to the computer on which you want to develop the client, for example the oimclient/ directory. This directory is referred to as *OIM_CLIENT_HOME* in this document. Extract the ZIP file. Note that the oimclient.zip file consists of the conf, lib, and oimclient.jar.
2. Copy the following files manually after unzipping oimclient.zip in the remote machine:
    a. Copy $*OIM_HOME*/server/client/oimWASClient.ear to the same location of oimclient.jar.
    b. Copy $*OIM_HOME*/server/client/config/xl.policy to the conf directory.
    c. Copy $*OIM_HOME*/server/client/config/authws.conf to the conf directory.
    d. Copy $*MW_HOME*/oracle_common/modules/oracle.jrf_11.1.1/jrf-api.jar to the lib directory.

3. Copy the application server-specific client library to the *OIM_CLIENT_HOME*/lib/ directory. For Oracle WebLogic Server, wlfullclient.jar is the client library. It is created in *MIDDLEWARE_HOME*/*WL_HOME*/server/lib/ directory, for example, /scratch/beahome/wlserver_10.3/server/lib/. Check if wlfullclient.jar is present. If not, then you must generate one by using the jarbuilder tool. See Oracle WebLogic Server documentation on how to generate wlfullclient.jar.
4. If your client application uses any one of the following, then in addition to adding the oimclient.jar, add the *OIM_ORACLE_HOME*/modules/oracle.idm.ipf_11.1.2/ipf.jar file to your custom client applications:

   oracle.iam.passwordmgmt.vo.Challenge

   oracle.iam.passwordmgmt.vo.OimPasswordPolicy

   oracle.iam.passwordmgmt.vo.OimUserInfo

   oracle.iam.passwordmgmt.vo.OimUserStatus
5. Pass the following system properties for running API clients:
   - java.security.auth.login.config=OIM_CLIENT_HOME/conf/authwl.conf
   - APPSERVER_TYPE=wls
6. Make sure following jars are in the class path:
   - commons-logging.jar
   - spring.jar
   - oimclient.jar
   - wlfullclient.jar
   - jrf-api.jar
   - ipf.jar (if used as described in step 4)

20.5 Working With Legacy Oracle Identity Manager APIs

This section describes the following topics:

- Using a Result Set Object
- Handling Oracle Identity Manager Exceptions
- Cleaning Up

20.5.1 Using a Result Set Object

Legacy Oracle Identity Manager APIs extensively use the tcResultSet interface. The `Thor.API.tcResultSet` interface is a data structure that stores records retrieved from the database. Methods in the Oracle Identity Manager API that must return a set of data use a result set. This is a two-dimensional data structure in which the columns correspond to the attributes and rows correspond to the entities. For example, a result set that is returned by the method that searches for users, each row would represent data pertaining to one user, and each column in the row would be an attribute for that user.

You can scroll through the result set and retrieve individual entries corresponding to particular attributes by using the various methods provided. To locate a particular row in the result set, use the `goToRow()` method with the row number as a parameter. To retrieve the values for the columns from a row, use appropriate accessor methods, such as `getStringValue()`. To obtain the value from a specific column, pass the column name as a parameter to the accessor method. The column name is the descriptive code defined in the Oracle Identity Manager Meta-Data system.

The following table shows some sample metadata values. This mapping is based on lookup codes and can be looked up in the Design Console by using the Lookup Definition Form.

Column Code	Explanation
IT Resources.Name	The name of an IT resource
Process Definition.Name	The name of a provisioning process

Note: Keep track of the result set objects that are retrieved, because they will be required when updating an existing record.

The following is an example of how to use a result set. This example obtains a result set by calling the `findAllUsers()` method. This method searches for all users matching certain criteria:

```
tcResultSet moResultSet = moUserUtility.findAllUsers(mhAttribs);
```

To check if the `findAllUsers()` method returned any records, use the `isEmpty()` method, for example:

```
boolean mbEmpty = moResultSet.isEmpty();
```

To retrieve the number of records found, use the `getRowCount()` method. If no records are found, then the method returns `0`. The following is an example:

```
int mnNumRec = moResultSet.getRowCount();
```

To select a particular record in the system, use the `goToRow()` method:

```
moResultSet.goToRow(5);
```

To retrieve the values of attributes from the current row, use the appropriate accessor method, for example:

```
String msUserLastName = moResultSet.getStringValue("Users.Last Name");
```

20.5.2 Handling Oracle Identity Manager Exceptions

The API methods throw Oracle-defined Java exceptions. Instead of using the `getMessage()` method on the exception object received, you can access the `isMessage` internal variable to retrieve the exception message.

20.5.3 Cleaning Up

The `tcUtilityFactory` class manages all resources used by a utility or factory instance and provides a means to release these resources after they are used.

If you instantiate and use `tcUtilityFactory` to obtain utility class instances, to release the resources that are associated with the utility class, call the `close(utility Object)` method on the factory class. If the session has ended, then call the `close()` method on the factory instance to release all the utility classes, the session objects, and the database objects.

If you obtain a utility class directly by using static calls, after the utility object is no longer needed, call the `close(object)` method on the utility object.

20.6 Code Samples

This section contains the following code samples:

- Retrieving Oracle Identity Manager Information
- Using Certification APIs
- Using OIMService API

20.6.1 Retrieving Oracle Identity Manager Information

Example 20–1 illustrates how to retrieve Oracle Identity Manager information. This example creates an instance of the factory class. The instance is then called several times to retrieve individual utility classes and use them to retrieve Oracle Identity Manager information.

Example 20–1 Retrieving Oracle Identity Manager Information

```
/*
 This class is intented to showcase some of OIM API's. These API's are
 specific to OIM 11g release. As an example, Legacy API's usage for
 Organization is also shown.
*/


package oracle.iam.samples;


// Role related API's
import oracle.iam.identity.rolemgmt.api.RoleManager;
import oracle.iam.identity.rolemgmt.vo.Role;
import oracle.iam.identity.exception.RoleSearchException;
import oracle.iam.identity.rolemgmt.api.RoleManagerConstants.RoleAttributeName;
import oracle.iam.identity.rolemgmt.api.RoleManagerConstants.RoleCategoryAttributeName;

// User related API's
import oracle.iam.identity.usermgmt.api.UserManager;
import oracle.iam.identity.usermgmt.vo.User;
import oracle.iam.identity.exception.UserSearchException;
import oracle.iam.identity.usermgmt.api.UserManagerConstants.AttributeName;

// Organization Legacy API's
import Thor.API.Operations.tcOrganizationOperationsIntf;
import Thor.API.tcResultSet;
import Thor.API.Exceptions.tcAPIException;
import Thor.API.Exceptions.tcColumnNotFoundException;
import Thor.API.Exceptions.tcOrganizationNotFoundException;

import oracle.iam.platform.OIMClient;
import oracle.iam.platform.authz.exception.AccessDeniedException;
import oracle.iam.platform.entitymgr.vo.SearchCriteria;

import java.util.*;
import javax.naming.NamingException;
import javax.security.auth.login.LoginException;


public class Sample {

     private static OIMClient oimClient;

     /*
      * Initialize the context and login with client supplied environment
     */
     public void init() throws LoginException {
        System.out.println("Creating client....");
        String ctxFactory = "weblogic.jndi.WLInitialContextFactory";
        String serverURL = "t3://OIM_HOSTNAME:OIM_PORT";
        String username = "xelsysadm";
        char[] password = "xelsysadm".toCharArray();
        Hashtable env = new Hashtable();
        env.put(OIMClient.JAVA_NAMING_FACTORY_INITIAL,ctxFactory);
        env.put(OIMClient.JAVA_NAMING_PROVIDER_URL, serverURL);

        oimClient = new OIMClient(env);
        System.out.println("Logging in");
        oimClient.login(username, password);
        System.out.println("Log in successful");
     }

     /**
     * Retrieves User login based on the first name using OIM 11g
     * UserManager service API.
     */
     public List getUserLogin(String psFirstName) {
        Vector mvUsers = new Vector();
        UserManager userService = oimClient.getService(UserManager.class);
        Set<String> retAttrs = new HashSet<String>();

        // Attributes that should be returned as part of the search.
        // Retrieve "User Login" attribute of the User.
        // Note: Additional attributes can be specified in a
        // similar fashion.
        retAttrs.add(AttributeName.USER_LOGIN.getId());

        // Construct a search criteria. This search criteria states
        // "Find User(s) whose 'First Name' equals 'psFirstName'".
        SearchCriteria criteria;
        criteria = new SearchCriteria(AttributeName.FIRSTNAME.getId(), psFirstName,
SearchCriteria.Operator.EQUAL);
        try {
            // Use 'search' method of UserManager API to retrieve
            // records that match the search criteria. The return
            // object is of type User.
            List<User> users = userService.search(criteria, retAttrs, null);

            for (int i = 0; i < users.size(); i++) {
                //Print User First Name and Login ID
                System.out.println("First Name : " + psFirstName + "  --  Login ID : " +
users.get(i).getLogin());
                mvUsers.add(users.get(i).getLogin());
            }
        } catch (AccessDeniedException ade) {
            // handle exception
        } catch (UserSearchException use) {
            // handle exception
        }
       return mvUsers;
    }

    /**
     * Retrieves the administrators of an Organization based on the
     * Organization name. This is Legacy service API usage.
     */
    public List getAdministratorsOfOrganization(String psOrganizationName) {
        Vector mvOrganizations = new Vector();
        tcOrganizationOperationsIntf moOrganizationUtility =
oimClient.getService(tcOrganizationOperationsIntf.class);
        Hashtable mhSearchCriteria = new Hashtable();
        mhSearchCriteria.put("Organizations.Organization Name", psOrganizationName);
        try {
            tcResultSet moResultSet = moOrganizationUtility.findOrganizations(mhSearchCriteria);
            tcResultSet moAdmins;
            for (int i = 0; i < moResultSet.getRowCount(); i++) {
                moResultSet.goToRow(i);
                moAdmins =
moOrganizationUtility.getAdministrators(moResultSet.getLongValue("Organizations.Key"));
                mvOrganizations.add(moAdmins.getStringValue("Groups.Group Name"));
                System.out.println("Organization Admin Name : " +
moAdmins.getStringValue("Groups.Group Name"));
            }
        } catch (tcAPIException tce) {
            // handle exception
        } catch (tcColumnNotFoundException cnfe) {
            // handle exception
        } catch (tcOrganizationNotFoundException onfe) {
            // handle exception
        }
        return mvOrganizations;
    }

    /**
     * Retrieves Role Display Name based on Role name and Role Category
     * using OIM 11g RoleManager service API. This example shows how
     * to construct compound search criteria.
     */
    public List getRoleDisplayName(String roleName, String roleCategory ) {
        Vector mvRoles = new Vector();
        RoleManager roleService = oimClient.getService(RoleManager.class);
        Set<String> retAttrs = new HashSet<String>();

        // Attributes that should be returned as part of the search.
        // Retrieve the "Role Display Name" attribute of a Role.
        // Note: Additional attributes can be specified in a
        // similar fashion.
        retAttrs.add(RoleAttributeName.DISPLAY_NAME.getId());

        // Construct the first search criteria. This search criteria
        // states "Find Role(s) whose 'Name' equals 'roleName'".
        SearchCriteria criteria1;
        criteria1 = new SearchCriteria(RoleAttributeName.NAME.getId(), roleName,
SearchCriteria.Operator.EQUAL);
```

```
        // Construct the second search criteria. This search criteria
        // states "Find Role(s) whose 'category' equals 'roleCategory'".
        SearchCriteria criteria2;
        criteria2 = new SearchCriteria(RoleCategoryAttributeName.NAME.getId(), roleCategory,
SearchCriteria.Operator.EQUAL);

        // Construct the compound search criteria using 'criteria1' and
        // 'criteria2' as arguments. This showcases how to construct
        // compound search criterias.
        SearchCriteria criteria = new SearchCriteria(criteria1, criteria2,
SearchCriteria.Operator.AND);
        try {
            // Use 'search' method of RoleManager API to retrieve
            // records that match the search criteria. The return
            // object is of type Role.
            List<Role> roles = roleService.search(criteria, retAttrs, null);

            for (int i = 0; i < roles.size(); i++) {
                //Print Role Display Name
                System.out.println("Role Display Name : " +
                    roles.get(i).getDisplayName());
                mvRoles.add(roles.get(i).getDisplayName());
            }
        } catch (AccessDeniedException ade) {
            // handle exception
        } catch (RoleSearchException use) {
            // handle exception
        }
        return mvRoles;
    }

    // Main method invocation
    // Following assumptions are made
    //1. A User "Joe Doe" already exists in OIM
    //2. An Organization  "Example Organization" already exists in OIM
    //3. A Role "Foobar" already exists in OIM
    public static void main(String args[]) {
        List moList = null;

        try {
            Sample oimSample = new Sample();

            // initialize resources
            oimSample.init();
            // retrieve User logins with first name 'Joe'
            moList=oimSample.getUserLogin("Joe");
            // retrieve User logins with first names starting with 'J'
            moList=oimSample.getUserLogin("J*");
            // retrieve the administrators of an Organization with name
            // 'Example Organization'
            moList=oimSample.getAdministratorsOfOrganization(
                "Example Organization");
            // retrieve Role display name with role name 'FooBar'
            // and role category as 'Defaut'
            moList=oimSample.getRoleDisplayName("foobar", "Default");
            // release resources
            oimClient.logout();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

The following is the sample output:

```
[java] Creating client....
[java] Logging in
[java] Log in successful
[java] First Name : Joe  --  Login ID : JDOE
[java] First Name : J*  --  Login ID : JHOND
[java] First Name : J*  --  Login ID : JDOE
[java] Organization Admin Name : SYSTEM ADMINISTRATORS
[java] Role Display Name : foobar
```

20.6.2 Using Certification APIs

This section provides code examples for using APIs related to certification, such as `CertificationService`.

Example 20–2 provides the code sample to get certifications belonging to a user.

Example 20–2 Retrieving Certifications Belonging to a User

```
public List<CertificationInstance> findCertifications(SearchCriteria
searchCriteria, Set<String> retAttrs, Map<String,Object> configParams) throws
CertificationServiceException;
Example of searchCriteria to use:
  SearchCriteria searchCriteria1 = new
SearchCriteria(CertificationConstants.CERTIFICATION_SEARCH_FIELDPRIMARY_REVIEW
ER_ID, userKey, SearchCriteria.Operator.EQUAL);
  SearchCriteria searchCriteria2 = new
SearchCriteria("certificationStatusForQuery",
CertificationConstants.STATE_IN_PROGRESS.toString(),
SearchCriteria.Operator.EQUAL);
  SearchCriteria searchCriteria = new SearchCriteria(searchCriteria1,
searchCriteria2, SearchCriteria.Operator.AND);
```

Example 20–3 provides the code sample to retrieve an application instance certification.

Example 20–3 Retrieving an Application Instance Certification

```
public List<IDCAccountAttributeAndRoleWrapper> loadBatchUserEntitlements(Long
certificationId, String taskUid, Long userId, PaginationContext context,
SearchCriteria searchCriteria) throws CertificationServiceException;
```

Example 20–4 provides the code sample to certify or deny entitlements.

Example 20–4 Certify or Deny Certifications

```
public void certifyUserEntitlements(Long certId, String taskUid, Long
userEntityId, Set<Long> roleEntityIds, Set<Long> accountEntityIds, Set<Long>
accountAttributeEntityIds, Integer certified, Date statusEndDate, String
comments) throws CertificationServiceException;
```

Example 20–5 provides the code sample to complete the certification.

Example 20–5 Complete the Certification

```
public CertificationInstance completeCertification(final Long certificationId,
String taskUid, char[] cleartextPassword)
```

20.6.3 Using OIMService API

This section lists out sample usage for few operations by using the OIMService API. It contains the following topics:

- RequestData Object Construction
- Samples of OIMService API Usage

20.6.3.1 RequestData Object Construction

For invoking the operations supported through OIMService API, you must first construct the RequestData object.

For all the operations that involve target user and cart item, such as role, application instance, or entitlement, construct the RequestData object as follows:

1. Create an instance of the RequestData object.
2. Create List of Beneficiary object(s), set the fields, and associate the object with RequestData object by invoking:

   ```
   requestData.setBeneficiaries();
   ```

3. Create List of RequestBeneficiaryEntity object(s), and set the cart item data, such as entity type, entity key, and operation. Associate the object with Beneficiary object by invoking:

   ```
   beneficiary.setTargetEntities();
   ```

   The entityKey, which is set by using the `entity.setEntityKey()` method for a given operation, must be based on Table 20–3.

Table 20–3 Operation and entityKey

Operation	entityKey
Provision Application Instance	Application Instance Key
Modify Account	Account Key
Revoke Account	Account Key
Enable Account	Account Key
Disable Account	Account Key
Provision Entitlement	Entitlement Key
Modify Entitlement	Entitlement Instance Key
Revoke Entitlement	Entitlement Instance Key

4. Create List of RequestBeneficiaryEntityAttribute object(s), and set the attribute name and value in each object. Associate the object with the entity object by invoking `entity.setEntityData()`.This is required only if the cart item is associated with a form.

Note:

- RequestData.setTargetEntities() must not be used in this scenario.
- See ""Samples of OIMService API Usage" on page 20-14" for more details.

For all the operations that involve only the User entity, such as Create User, Modify User, Enable User, Disable User, and Delete User, the RequestData object must be populated as follows:

1. Create an instance of the RequestData object.
2. Create List of RequestEntity object(s) by setting entity type, entity key, and operation. Entity key must be set as user key for all the operations exception Create User.
3. Create List of RequestEntityAttribute object(s), and set attribute name and value in each object. This is required only for Create User and Modify User operations.

20.6.3.2 Samples of OIMService API Usage

This section provides code samples of using the OIMService API.

Example 20–6 provides the code sample for revoking an account.

Example 20–6 Revoking an Account

```
RequestData requestData = new RequestData();
Beneficiary beneficiary = new Beneficiary();
beneficiary.setBeneficiaryKey("12"); //User with key 12
beneficiary.setBeneficiaryType(Beneficiary.USER_BENEFICIARY);

RequestBeneficiaryEntity entity = new RequestBeneficiaryEntity();
entity.setEntityType("ApplicationInstance");
entity.setEntityKey(String.valueOf(accountKey));
entity.setOperation("REVOKE");

List<RequestBeneficiaryEntity> entities = new
ArrayList<RequestBeneficiaryEntity>();

entities.add(entity);
beneficiary.setTargetEntities(entities);

List<Beneficiary> beneficiaries = new ArrayList<Beneficiary>();
beneficiaries.add(beneficiary);
requestData.setBeneficiaries(beneficiaries);
OperationResult result = oimService.doOperation(requestData,
OIMService.Intent.ANY);
if( result.getRequestID() != null ) {
//Operation resulted in to request creation.
System.out.println("Request submitted with ID: " + result.getRequestID());
} else {
System.out.println("Account is revoked successfully");
}
```

Example 20–7 provides the code sample for creating a user.

Example 20–7 Creating a User

```
RequestData requestData = new RequestData("Create User");
RequestEntity ent = new RequestEntity();
ent.setRequestEntityType(OIMType.User);
ent.setOperation("CREATE");
HashMap<String, String> userData = new HashMap<String, String>();

List<RequestEntityAttribute> attrs = new ArrayList<RequestEntityAttribute>();

RequestEntityAttribute attr = new RequestEntityAttribute("Last Name", "Doe",
RequestEntityAttribute.TYPE.String);
attrs.add(attr);
attr = new RequestEntityAttribute("First Name", "John",
RequestEntityAttribute.TYPE.String);
attrs.add(attr);
attr = new RequestEntityAttribute("User Login", "jdoe",
RequestEntityAttribute.TYPE.String);
attrs.add(attr);
Long organizationKey = new Long(1);
attr = new RequestEntityAttribute("Organization", organizationKey ,
RequestEntityAttribute.TYPE.Long);
attrs.add(attr);
attr = new RequestEntityAttribute("Role", "Full-Time",
RequestEntityAttribute.TYPE.String);
attrs.add(attr);

ent.setEntityData(attrs);

List<RequestEntity> entities = new ArrayList<RequestEntity>();
entities.add(ent);
requestData.setTargetEntities(entities);
OperationResult result = oimService.doOperation(requestData,
OIMService.Intent.ANY);
if( result.getRequestID() != null ) {
//Operation resulted in to request creation.
System.out.println("Request submitted with ID: " + result.getRequestID());
} else {
System.out.println("User is created successfully");
}
```

21 Using SCIM/REST Services

Representation State Transfer (REST) is an architectural style for building web services over HTTP. Identity REST services are a set of REST web services that provide functionality for self-service, user, role/group, organization, and password policy management. Identity REST services are based on the System for Cross-Domain Identity Management (SCIM) protocol. Oracle Identity Manager SCIM service is available by default with the SCIM schema and IDM extensions, as described in "Schema Attributes for the User Resource" on page 21-4.

The supported schema can be retrieved, as described in "Retrieving Schemas" on page 21-73.

When you deploy Oracle Identity Manager, SCIM is deployed by default as a web application on the Oracle Identity Manager server.

SCIM implementation in Oracle Identity Manager follows draft-ietf-scim-api-13 and draft-ietf-scim-core-schema-13. For information about IETF drafts, refer to the following URL:

`http://www.simplecloud.info/`

This chapter contains the following topics:

- Supported Resources and Operations
- Resource Schema
- Operation Types
- HTTP Response Codes
- SCIM-Based API Examples
- Securing SCIM Resources

21.1 Supported Resources and Operations

Table 21–1 lists SCIM-based APIs used for the supported operations in Oracle Identity Manager.

Table 21–1 SCIM-Based APIs and Supported Operations

Resource	Endpoint	Operation	Schema URL	Description
User	/Users	GET, POST, PUT, PATCH, DELETE	urn:ietf:params:scim:schemas:core:2.0:User urn:ietf:params:scim:schemas:extension:enterprise:2.0:User . urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:User . urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:User	Get/Add/Modify/Disable/Enable/Lock/Unlock/Delete Users - identity.usermgmt.api.UserManager
User	/Me	GET, POST, PUT, PATCH	·urn:ietf:params:scim:schemas:core:2.0:User . urn:ietf:params:scim:schemas:extension:enterprise:2.0:User . urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:User . urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:User	Get/Modify My Profile, Change My Password, Change My Challenge Responses -, selfservice.self.selfmgmt.api.AuthenticatedSelfService Self Registration - via UnauthenticatedSelfService
PasswordResetterWithChallenges	/PasswordResetterWithChallenges	POST	· urn:ietf:params:scim:schema:oracle:core:2.0:PasswordResetterWithChallenges	
PasswordValidator	/PasswordValidator	POST	· urn:ietf:params:scim:schema:oracle:core:2.0:PasswordValidator	
UserNameGenerator	/UserNameGenerator	POST	· urn:ietf:params:scim:schema:oracle:core:2.0:UserNameGenerator	
UserNameRecoverer	/UserNameRecoverer	POST	· urn:ietf:params:scim:schema:oracle:core:2.0:UserNameRecoverer	
UserNameValidator	/UserNameValidator	POST	· urn:ietf:params:scim:schema:oracle:core:2.0:UserNameValidator	

Table 21–1 (Cont.) SCIM-Based APIs and Supported Operations

Resource	Endpoint	Operation	Schema URL	Description
Group	/Groups	GET, POST, PUT, PATCH, DELETE	· urn:ietf:params:scim:schemas:core:2.0:Group · urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:Group · urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:Group	Get/Add/Modify/SetUserMembershipRule/Delete Groups - identity.rolemgmt.api.RoleManager
Organization	/Organizations	GET, POST, PUT, PATCH, DELETE	urn:ietf:params:scim:schemas:oracle:core:2.0:OIG:Organization	Get/Add/Modify/SetUserMembershipRule/Delete Organizations - identity.orgmgmt.api.OrganizationManager
Password Policy	/PasswordPolicies	GET, POST, PUT, PATCH, DELETE	urn:ietf:params:scim:schemas:oracle:core:2.0:IDM:PasswordPolicy	Get/Add/Modify/Delete Password Policies - passwordmgmt.api.PasswordMgmtService
Notification Template	/NotificationTemplates	GET, POST, PUT, PATCH, DELETE	urn:ietf:params:scim:schemas:oracle:core:2.0:OIG:NotificationTemplate	Get/Add/Modify/Delete Notification Templates - notification.api.NotificationService
System Property	/SystemProperties	GET, PATCH	urn:ietf:params:scim:schemas:oracle:core:2.0:OIG:SystemProperty	Get/Modify System Property - config.api.SystemConfigurationService
Service Provider Configuration Schema	/ServiceProviderConfigs	GET	urn:ietf:params:scim:schemas:core:2.0:ServiceProviderConfig	Get the service provider's configuration
Resource Type	/ResourceTypes	GET	urn:ietf:params:scim:schemas:core:2.0:ResourceType	Get the resource type's configuration
Schema	/Schemas	GET	urn:ietf:params:scim:schemas:core:2.0:Schema urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:Schema	Get a resource's schema
Search	[prefix]/.search	POST	NA	Perform search at system root or with in a resource endpoint for one or more resource types using POST

21.2 Resource Schema

The resource schema tables listed in this section show the supported SCIM attributes. All SCIM resource types and schema extensions are identified by the following URI in both JSON requests and responses:

`urn:oracle:scim:schemas:idm:2.0:`*`RESOURCE_TYPE`*

All SCIM resources, such as users, groups, and organizations, include the following types of SCIM schema attributes:

- **SVA:** Single-valued attribute
- **MVA:** Multi-valued attribute.

- **CSVA:** Complex single-valued attribute
- **CMVA:** Complex multi-valued attribute.

  SCIM user schema supports CMVA, such as email address, where each value can have subattributes, such as personal email address, work email address, and other email address, and value. As Oracle Identity Manager does not support CMVA, Oracle Identity Manager SCIM/REST API also does not support them, except where they can be mapped to existing Oracle Identity Manager user schema attributes. If a request is made that includes a complex SCIM attribute that is not supported by the Oracle Identity Manager SCIM/REST, then an error is returned in the REST response indicating the same.

Mutability is the way a given attribute is accessed. The possible mutability values are:

- Read-only (RO): Allows create and read/search operations
- WO: Allows create but not read/search operations
- RW: Allows create as well as read/search operations

21.2.1 Schema Attributes for the User Resource

Table 21–2 lists the SCIM user schema attributes.

Table 21–2 urn:ietf:params:scim:schemas:core:2.0:User

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
userName	SVA	String	Y	RW
name.formatted	SVA	String	N	RW
name.familyName	SVA	String	N	RW
name.givenName	SVA	String	N	RW
name.middleName	SVA	String	N	RW
name.honorificPrefix	SVA	String	N	RW
name.honorificSuffix	SVA	String	N	RW
displayName	SVA	String	N	RW
nickName	SVA	String	N	RW
profileUrl	SVA	String	N	RW
title	SVA	String	N	RW
title	SVA	String	N	RW
userType	SVA	String	N	RW
preferredLanguage	SVA	String	N	RW
timezone	SVA	String	N	RW
locale	SVA	String	N	RW
active	SVA	Boolean	N	RW
password	SVA	String	N	WO
emails	CMVA	NA	N	RW
emails[work].value	SVA	String	N	RW
emails[<type>].primary	SVA	String	N	RW

Table 21–2 (Cont.) urn:ietf:params:scim:schemas:core:2.0:User

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
phoneNumbers	CMVA	NA	N	RW
phoneNumbers.type	SVA	String	N	RW
phoneNumbers[work].value	SVA	String	N	RW
phoneNumbers[home].value	SVA	String	N	RW
phoneNumbers[mobile].value	SVA	String	N	RW
phoneNumber[fax].value	SVA	String	N	RW
phoneNumber[pager].value	SVA	String	N	RW
phoneNumber[other].value	SVA	String	N	RW
ims	CMVA	NA	N	RW
photos	CMVA	NA	N	RW
addresses	CMVA	NA	N	RW
addresses.type	SVA	String	N	RW
addresses[<type>].primary	SVA	String	N	RW
addresses[work].formatted	SVA	String	N	RW
addresses[home].formatted	SVA	String	N	RW
addresses[work].streetAddress	SVA	String	N	RW
addresses[work].locality	SVA	String	N	RW
addresses[work].region	SVA	String	N	RW
addresses[work].postalCode	SVA	String	N	RW
addresses[work].country	SVA	String	N	RW
groups	CMVA	NA	N	RO
groups.value	SVA	String	N	RO
groups.$ref	SVA	String	N	RO
groups.type	SVA	String	N	RO
entitlements	SMVA	String	N	RW
roles	SMVA	String	N	RW
x509Certificates	SMVA	String	N	RW

Note: Accounts and entitlements are not supported by Oracle Identity Manager SCIM services.

Table 21–3 lists the SCIM enterprise user schema attributes.

Table 21–3 urn:ietf:params:scim:schemas:extension:enterprise:2.0:User

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
employeeNumber	SVA	String	N	RW

Table 21–3 (Cont.) urn:ietf:params:scim:schemas:extension:enterprise:2.0:User

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
costCenter	SVA	String	N	RW
organization	SVA	String	N	RO
division	SVA	String	N	RW
department	SVA	String	N	RW
manager.value	SVA	String	N	RW
manager.$ref	SVA	String	N	RW
manager.displayName	SVA	String	N	RO

Table 21–4lists the SCIM IDM common user schema extension attributes.

Table 21–4 urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:User

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
createBy.value	SVA	String	N	RO
createBy.$ref	SVA	String	N	RO
updateBy.value	SVA	String	N	RO
updateBy.$ref	SVA	String	N	RO
passwd	CSVA			
passwd.value	SVA	String	N	WO
passwd.oldValue	SVA	String	N	WO
passwd.sendNotification	SVA	String	N	WO
passwd.sendNotificationTo	SVA	String	N	WO
passwordMustChange	SVA	String	N	RO
passwordExpireDate	SVA	String	N	RO
locked.value	SVA	String	N	RW
locked.duration	SVA	String	N	RW
locked.reason	SVA	String	N	RO
locked.on	SVA	String	N	RO
challenges	CMVA	NA	N	RW
challenges.challenge	SVA	String	N	RW
challenges.response	SVA	String	N	RW

Table 21–5 lists the Oracle Identity Governance (OIG) user schema extension attributes.

Table 21–5 urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:User

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
dataLevel	SVA	String	N	RO
disabled	SVA	String	N	RO
passwordCreateDate	SVA	Date	N	RO
passwordCantChange	SVA	String	N	RO
passwordNeverExpires	SVA	String	N	RO
passwordIsExpired	SVA	String	N	RO
passwordWarnDate	SVA	Date	N	RO
lastSuccessfulLoginDate	SVA	Date	N	RO
lastFailedLoginDate	SVA	Date	N	RO
hireDate	SVA	Date	N	RW
startDate	SVA	Date	N	RW
endDate	SVA	Date	N	RW
provisioningDate	SVA	Date	N	RW
provisionedDate	SVA	Date	N	RO
deprovisioningDate	SVA	Date	N	RW
deprovisionedDate	SVA	Date	N	RO
automaticallyDeleteOn	SVA	Date	N	RO
userLoginAttemptsCounter	SVA	Int	N	RO
userPasswordResetAttemptsCounter	SVA	Int	N	RO
userMustChangePasswordAtNextLogin	SVA	String	N	RO
userPasswordMinAgeDate	SVA	Date	N	RO
description	SVA	String	N	RW
ldapCommonName	SVA	String	N	RW
ldapCommonNameGenerated	SVA	String	N	RW
ldapOrganization	SVA	String	N	RW
ldapOrganizationalUnit	SVA	String	N	RW
ldapDn	SVA	String	N	RW
ldapGuid	SVA	String	N	RW
poBox	SVA	String	N	RW
jobCode	SVA	String	N	RW
officeName	SVA	String	N	RW

Table 21–5 (Cont.) urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:User

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
initials	SVA	String	N	RW
faLanguage	SVA	String	N	RW
faTerritory	SVA	String	N	RW
embeddedHelp	SVA	String	N	RW
fontSize	SVA	String	N	RW
colorContrast	SVA	String	N	RW
accessibilityMode	SVA	String	N	RW
numberFormat	SVA	String	N	RW
dateFormat	SVA	String	N	RW
timeFormat	SVA	String	N	RW
currency	SVA	String	N	RW
summaryRisk	SVA	String	N	RO
hasHighRiskRole	SVA	String	N	RO
hasHighRiskResource	SVA	String	N	RO
hasHighRiskEntitlement	SVA	String	N	RO
hasHighRiskProvisioningMethod	SVA	String	N	RO
hasHighRiskOpenSod	SVA	String	N	RO
hasHighRiskLastCert	SVA	String	N	RO
roleSummaryRisk	SVA	String	N	RO
accountSummaryRisk	SVA	String	N	RO
entitlementSummaryRisk	SVA	String	N	RO
riskUpdateDate	SVA	String	N	RO
homeOrganization	CSVA	NA	N	RW
homeOrganization.value	SVA	String	N	RW
homeOrganization.$ref	SVA	String	N	RO
Organizations	CMVA	NA	N	RO
organizations.value	SVA	String	N	RO
organizations.$ref	SVA	String	N	RO
passwordPolicyDescription	SVA	String	N	RO
requestId	SVA	String	N	RO

21.2.2 Schema Attributes for the PasswordResetterWithChallenges Resource

Table 21–6 lists the IDM PasswordResetterWithChallenges user schema attributes.

Table 21–6 urn:ietf:params:scim:schemas:oracle:core:2.0:PasswordResetterWithChallenges

SCIM Attributes	Attribute Type	Data Type	Required	Mutability
userName	SVA	String	Y	WO
Challenges	CMVA	NA	Y	WO
challenges.challenge	SVA	String	Y	WO
challenges.response	SVA	String	Y	WO
password	SVA	String	Y	WO

21.2.3 Schema Attributes for the PasswordValidator Resource

Table 21–7 lists the IDM PasswordValidator schema attributes.

Table 21–7 urn:ietf:params:scim:schemas:oracle:core:2.0:PasswordValidator

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
$ref	SVA	String	Y	WO
password	SVA	String	Y	WO

21.2.4 Schema Attributes for the UserNameValidator Resource

Table 21–8 lists the IDM UserNameValidator schema attributes.

Table 21–8 urn:ietf:params:scim:schemas:oracle:core:2.0:UserNameValidator

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
userName	SVA	String	Y	WO

21.2.5 Schema Attributes for the UserNameGenerator Resource

Table 21–9 lists the IDM UserNameGenerator schema attributes.

Table 21–9 urn:ietf:params:scim:schemas:oracle:core:2.0:UserNameGenerator

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
name	CSVA	NA	Y	WO
name.formatted	SVA	String	Y	WO
name.familyName	SVA	String	Y	WO
name.givenName	SVA	String	Y	WO
name.middleName	SVA	String	Y	WO
name.honorificSuffix	SVA	String	Y	WO

21.2.6 Schema Attributes for the UserNameRecoverer Resource

Table 21–10 lists the IDM UserNameRecoverer schema attributes.

Table 21–10 urn:ietf:params:scim:schemas:oracle:core:2.0:UserNameRecoverer

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
email	SVA	String	Y	WO

21.2.7 Schema Attributes for the Group Resource

Table 21–11 lists the SCIM group schema attributes.

Table 21–11 urn:ietf:params:scim:schemas:core:2.0:Group

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
displayName	SVA	String	Y	RW
members	CMVA	NA	N	
members.value	SVA	String	N	RW
members .$ref	SVA	String	N	RW

Table 21–12 lists the IDM common group schema extension attributes.

Table 21–12 urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:Group

SCIM Attributes	Attribute Type	Data Type	Required	Mutability
createBy.value	SVA	String	N	RO
createBy.$ref	SVA	String	N	RO
updateBy.value	SVA	String	N	RO
updateBy.$ref	SVA	String	N	RO
email	SVA	String	N	RW
description	SVA	String	N	RW
owner	CSVA	NA	N	RW
owner.value	SVA	String	N	RW
owner.$ref	SVA	String	N	RO
owner.firstName	SVA	String	N	RO
owner.lastName	SVA	String	N	RO
owner.displayName	SVA	String	N	RO
owner.email	SVA	String	N	RO
owner.login	SVA	String	N	RO

Table 21–13 lists the OIG group schema extension attributes.

Table 21–13 urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:Group

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
dataLevel	SVA	String	N	RO
namespace	SVA	String	N	RW
category	CSVA	NA	N	RW
category.value	SVA	String	N	RW

Table 21–13 (Cont.) urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:Group

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
category.name	SVA	String	N	RO
ldapGuid	SVA	String	N	RO
ldapDn	SVA	String	N	RO
requestId	SVA	String	N	RO
accessPolicies.value	MVA	String	N	RW
organizationsPublishedTo	CMVA	NA	N	RW
organizationsPublishedTo.value	SVA	String	N	RW
organizationsPublishedTo.$ref	SVA	String	N	RO
catalog	CSVA	NA	N	RW
catalog.id	SVA	String	N	RO
catalog.categoryName	SVA	String	N	RW
catalog.auditObjectives	SVA	String	N	RW
catalog.itemRisk	SVA	Integer	N	RW
catalog.userDefinedTags	SVA	String	N	RW
catalog.certifiable	SVA	Boolean	N	RW
catalog.auditable	SVA	Boolean	N	RW
catalog.requestable	SVA	Boolean	N	RW
catalog.tags	SVA	String	N	RO
catalog.hierarchicalDataAvailable	SVA	Boolean	N	RO
catalogApproverUser.value	SVA	String	N	RW
catalogApproverUser.$ref	SVA	Reference	N	RW
catalogApproverRole.value	SVA	String	N	RW
catalogApproverRole.$ref	SVA	Reference	N	RW
catalogCertifierUser.value	SVA	String	N	RW
catalogCertifierUser.$ref	SVA	Reference	N	RW
catalogCertifierRole.value	SVA	String	N	RW
catalogCertifierRole.$ref	SVA	Reference	N	RW
catalogFulfillmentUser.value	SVA	String	N	RW

Table 21–13 (Cont.) urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:Group

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
catalogFulfillmentUser.$ref	SVA	Reference	N	RW
catalogFulfillmentRole.value	SVA	String	N	RW
catalogFulfillmentRole.$ref	SVA	Reference	N	RW
catalogAttributes	CMVA	NA	N	RW
catalogAttributes.name	SVA	String	N	RW
catalogAttributes.value	SVA	String	N	RW
catalogAttributes.udf	SVA	Boolean	N	RW
catalogAttributes.description	SVA	String	N	RW
catalogAttributes.searchable	SVA	Boolean	N	RW
catalogAttributes.sortable	SVA	Boolean	N	RW
catalogAttributes.certifiable	SVA	Boolean	N	RW
catalogAttributes.datatype	SVA	String	N	RO
userMembershipRule	CSVA	NA	N	RW
userMembershipRule.value	SVA	String	N	RW
userMembershipRule.evaluate	SVA	Boolean	N	WO

21.2.8 Schema Attributes for the Organization Resource

Table 21–14 lists the OIG organization schema attributes.

Table 21–14 urn:ietf:params:scim:schemas:oracle:core:2.0:OIG:Organization

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
createBy.value	SVA	String	N	RO
createBy.$ref	SVA	String	N	RO
updateBy.value	SVA	String	N	RO
updateBy.$ref	SVA	String	N	RO
dataLevel	SVA	String	N	RO
name	SVA	String	N	RW
customerType	SVA	String	N	RW
status	SVA	String	N	RW
disabled	SVA	String	N	RW
parent	CSVA	NA	N	RW

Table 21–14 (Cont.) urn:ietf:params:scim:schemas:oracle:core:2.0:OIG:Organization

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
parent.value	SVA	String	N	RW
parent.$ref	SVA	String	N	RO
parent.name	SVA	String	N	RO
passwordPolicy	CSVA	NA	N	RW
passwordPolicy.value	SVA	String	N	RW
passwordPolicy.$ref	SVA	String	N	RO
passwordPolicy.name	SVA	String	N	RO
certifierUser	CSVA	NA	N	RW
certifierUser.value	SVA	String	N	RW
certifierUser.$ref	SVA	String	N	RO
certifierUser.login	SVA	String	N	RO
enforceNewPasswordPolicy	SVA	String	N	RW
userMembershipRule	CSVA	NA	N	RW
userMembershipRule.value	SVA	String	N	RW
userMembershipRule.evaluate	SVA	String	N	WO
members	CMVA	NA	N	RO
members.value	SVA	String	N	RO
members.$ref	SVA	String	N	RO
childOrganizations	CSVA	NA	N	RO
childOrganizations.value	SVA	String	N	RO
childOrganizations.$ref	SVA	Reference	N	RO

21.2.9 Schema Attributes for the Password Policy Resource

Table 21–15 lists the IDM password policy schema attributes.

Table 21–15 urn:ietf:params:scim:schemas:oracle:core:2.0:IDM:PasswordPolicy

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
name	SVA	String	N	RW
description	SVA	String	N	RW
maxLength	SVA	String	N	RW
minLength	SVA	String	N	RW
minAlphas	SVA	String	N	RW
minNumerals	SVA	String	N	RW
minAlphaNumerals	SVA	String	N	RW
minSpecialChars	SVA	String	N	RW

Table 21–15 (Cont.) urn:ietf:params:scim:schemas:oracle:core:2.0:IDM:PasswordPolicy

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
maxSpecialChars	SVA	String	N	RW
minUpperCase	SVA	String	N	RW
minLowerCase	SVA	String	N	RW
minUniqueChars	SVA	String	N	RW
maxRepeatedChars	SVA	String	N	RW
startsWithAlphabet	SVA	String	N	RW
minUnicodeChars	SVA	String	N	RW
maxUnicodeChars	SVA	String	N	RW
firstNameDisallowed	SVA	String	N	RW
lastNameDisallowed	SVA	String	N	RW
userIdDisallowed	SVA	String	N	RW
minPasswordAgeInDays	SVA	String	N	RW
passwordWarningAfterInDays	SVA	String	N	RW
passwordExpiresAfterInDays	SVA	String	N	RW
requiredChars	SVA	String	N	RW
disallowedChars	SVA	String	N	RW
allowedChars	SVA	String	N	RW
disallowedSubstrings	SVA	String	N	RW
dictionaryLocation	SVA	String	N	RW
dictionaryDelimiter	SVA	String	N	RW
numPasswordsInHistory	SVA	String	N	RW
maxIncorrectAttempts	SVA	String	N	RW
lockoutDuration	SVA	String	N	RW
complexPolicy	SVA	String	N	RW
challengesEnabled	SVA	String	N	RW
challengeSource	SVA	String	N	RW
challengeDefaultQuestions.value	SVA	String	N	RW
challengeMinQuestions	SVA	String	N	RW
challengeMinAnswers	SVA	String	N	RW
challengeAllAtOnce	SVA	String	N	RW
challengeResponseMinLength	SVA	String	N	RW

Table 21–15 (Cont.) urn:ietf:params:scim:schemas:oracle:core:2.0:IDM:PasswordPolicy

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
challengeAllowDupli cateResponses	SVA	String	N	RW
challengeMaxIncorre ctAttempts	SVA	String	N	RW

21.2.10 Schema Attributes for the Notification Template Resource

Table 21–16 lists the OIG notification template schema attributes.

Table 21–16 urn:ietf:params:scim:schemas:oracle:core:2.0:OIG:NotificationTemplate

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
name	SVA	String	Y	RW
eventName	SVA	String	Y	RW
description	SVA	String	N	RW
locales	CMVA	NA	N	RW
locales.locale	SVA	String	Y	RW
locales.encoding	SVA	String	Y	RW
locales .subject	SVA	String	Y	RW
locales .contentType	SVA	String	Y	RW
locales.shortMessage	SVA	String	N	RW
locales.longMessage	SVA	String	Y	RW

21.2.11 Schema Attributes for the System Property Resource

Table 21–17 lists the OIG system property schema attributes.

Table 21–17 urn:ietf:params:scim:schemas:oracle:core:2.0:OIG:SystemProperty

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
name	SVA	String	Y	RW
displayName	SVA	String	N	RW
value	SVA	String	N	RW

21.2.12 Schema Attributes for the Service Provider Configuration Schema Resource

Table 21–18 lists the SCIM service provider configuration schema attributes.

Table 21–18 urn:ietf:params:scim:schemas:core:2.0:ServiceProviderConfig

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
documentationUrl	SVA	String	N	RO
patch.supported	SVA	Boolean	N	RO
bulk.supported	SVA	Boolean	N	RO
bulk.maxOperations	SVA	Integer	N	RO
bulk.maxPayloadSize	SVA	Integer	N	RO

Table 21–18 (Cont.) urn:ietf:params:scim:schemas:core:2.0:ServiceProviderConfig

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
filter.supported	SVA	Boolean	N	RO
filter.maxResults	SVA	Integer	N	RO
changePassword.supported	SVA	Boolean	N	RO
sort.supported	SVA	Boolean	N	RO
Etag.supported	SVA	Boolean	N	RO
authenticationSchemes.name	SVA	String	N	RO
authenticationSchemes.description	SVA	String	N	RO
authenticationSchemes.specUrl	SVA	String	N	RO
authenticationSchemes.documentationUrl	SVA	String	N	RO

21.2.13 Schema Attributes for the Resource Type Resource

Table 21–19 lists the SCIM resource type schema attributes.

Table 21–19 urn:ietf:params:scim:schemas:core:2.0:ResourceType

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
name	SVA	String	N	RO
description	SVA	String	N	RO
endpoint	SVA	String	N	RO
schema	SVA	String	N	RO
schemaExtensions.schema	SVA	String	N	RO
schemaExtensions.required	SVA	Boolean	N	RO

21.2.14 Schema Attributes for the Schema Resource

Table 21–20 lists the attributes of the SCIM schema.

Table 21–20 urn:ietf:params:scim:schemas:core:2.0:Schema

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
id	SVA	String	N	RO
name	SVA	String	N	RO
description	SVA	String	N	RO
attributes.name	SVA	String	N	RO
attributes.type	SVA	String	N	RO
attributes.multiValued	SVA	String	N	RO
attributes.description	SVA	String	N	RO

Table 21–20 (Cont.) urn:ietf:params:scim:schemas:core:2.0:Schema

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
attributes.readOnly	SVA	Boolean	N	RO
attributes.required	SVA	Boolean	N	RO
attributes.mutability	SVA	String	N	RO
attributes.returned	SVA	String	N	RO
attributes.uniqueness	SVA	String	N	RO
attributes.caseExact	SVA	Boolean	N	RO

Table 21–21 lists the schema extension attribute of the OIG schema.

Table 21–21 urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:Schema

SCIM Attribute	Attribute Type	Data Type	Required	Mutability
attributes.indexes	SVA	String	N	RO

21.3 Operation Types

The following operation types are supported:

- **GET:** Retrieves one or more complete or partial resources.
- **POST:** Creates new resources or creates search requests, depending on the endpoint.
- **PUT:** Modifies a resource by replacing existing attributes with a specified set of replacement attributes (replace). PUT must not be used to create new resources.
- **PATCH:** Modifies a resource with a set of client-specified changes (partial updates).
- **DELETE:** Deletes a resource.

For more information about operation types, refer to the following URL:

`https://tools.ietf.org/html/draft-ietf-scim-api-13#page-5`

21.4 HTTP Response Codes

In addition to returning a HTTP response code, Identity REST services return the errors in the body of the response with error code and descriptions. Table 21–22 lists the error codes and their meaning.

Table 21–22 Error Codes and Meaning

Error Condition	HTTP Return Code	Meaning
Not able to parse input, input does not match required entities, or validation failures	400	Bad Request: validation failures, schema violations
Requested resource not found	404	Not found *ADDITIONAL_INFORMATION_INDICATING_NOT_FOUND_OBJECT*
User not authorized to execute service	401	Unauthorized

Table 21–22 (Cont.) Error Codes and Meaning

Error Condition	HTTP Return Code	Meaning
Requested method not supported	501	Method not allowed
Client does not accept produced content type	406	Not acceptable
Incorrect request parameter semantics	422	Unprocessable Entity. *ADDITIONAL_INFORMATION_ON _NATURE_OF_ERROR*
Client media type unsupported	415	Unsupported media type
Failed Dependency	424	Failed Dependency. *ADDITIONAL_INFORMATION_ON _FAILED_DEPENDENCY*
Generic server failure	500	Internal server error
conflict	409	The specified version number does not match, or the resource's latest version number or a service provider refused to create a new, duplicate resource
precondition failed	412	Failed to update as resource ID changed on the server last retrieved
forbidden	403	Server does not support requested operation on a given resource

Table 21–23 lists the success codes and their meaning.

Table 21–23 Success Codes and Meaning

HTTP Return Code	Meaning
200	Processed successfully.
201	The request has been fulfilled and resulted in a new resource being created.
204	The server has fulfilled the request but does not return a response body.

21.5 SCIM-Based API Examples

This section provides the following examples for SCIM-based API usage:

- User Management
- Role Management
- Organization Management
- Password Policy Management
- Notification Template Management
- System Property Management
- Service Provider Configuration Management
- Resource Types Management
- Using POST Search

- Retrieving Schemas

Note: You can use user defined fields (UDFs) in SCIM requests. After UDFs are created in Oracle Identity Manager, they automatically appear in SCIM resources as regular attributes. There is no difference in the requests and responses with regular attributes.

21.5.1 User Management

This section provides the following examples of the User resource:

- Create User
- Modify User (PUT)
- Modify User (PATCH)
- View Users with Pagination
- Delete User
- Lock User
- Unlock User
- Reset Password by Providing New Password
- Reset Password by Auto-Generated Password
- View User
- Self Registration
- Modify Self Profile (PATCH)
- Modify Profile (PUT)
- PasswordResetterWithChallenges
- PasswordValidator
- UserNameValidator
- UserNameGenerator
- UserNameRecoverer

21.5.1.1 Create User

Request:

- **Operation and URI:** `POST http://`*`HOST_NAME`*`:`*`PORT`*`/idaas/im/scim/v1/Users`
- **Header:**
  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`
- **Body:**

```
{
  "schemas":
  [
    "urn:ietf:params:scim:schemas:core:2.0:User",
    "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User",
    "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:User"
  ],
  "userName": "bjensen@example.com",
  "name": {
    "familyName": "Jensen",
    "givenName": "Barbara",
    "middleName": "Jane",
    "honorificSuffix": "III"
  },
  "displayName": "Babs Jensen",
  "profileUrl": "https://HOST_NAME:PORT/bjensen",
  "emails":
  [
    {
      "value": "bjensen@example.com",
      "type": "work"
    }
  ],
  "addresses": [
    {
      "type": "work",
      "streetAddress": "100 Universal City Plaza",
      "locality": "Hollywood",
      "region": "CA",
      "postalCode": "91608",
      "country": "USA",
      "formatted": "100 Universal City Plaza\nHollywood, CA 91608 USA"
    },
    {
      "type": "home",
      "formatted": "456 Hollywood Blvd\nHollywood, CA 91608 USA"
    }
  ],
  "phoneNumbers": [
    {
      "value": "555-555-5555",
      "type": "work"
    },
    {
      "value": "555-555-4444",
      "type": "mobile"
    }
  ],
  "userType": "Contractor",
  "title": "Tour Guide",
  "preferredLanguage":"en-US",
  "locale": "en-US",
  "timezone": "America/Los_Angeles",
  "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User":
  {
    "employeeNumber": "701984",
    "costCenter": "4130",
    "division": "Theme Park",
    "department": "Tour Operations",
    "manager":
    {
      "value": "1",
      "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
    }
  },
  "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:User":
  {
    "homeOrganization":
    {
      "value": "1",
      "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/1"
    }
  }
}
```

Response:

- **Status:** `HTTP/1.1 201 Created`
- **Body:**

```
{
    "schemas":
    [
       "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:User",
       "urn:ietf:params:scim:schemas:core:2.0:User",
       "urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:User",
       "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User"
    ],
    "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:User": {
        "userLoginAttemptsCounter": 0,
        "passwordIsExpired": "0",
        "ldapCommonNameGenerated": 0,
        "userPasswordResetAttemptsCounter": 0,
        "passwordWarnDate": "2015-04-29T03:24:16.000-07:00",
        "homeOrganization": {
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/1"
        },
        "passwordCreateDate": "2015-01-06T03:24:16.000-08:00",
        "provisionedDate": "2015-01-06T03:24:16.000-08:00",
        "passwordPolicyDescription": [],
        "userMustChangePasswordAtNextLogin": "1",
        "disabled": false,
        "organizations": [
            {
                "value": "1",
                "$ref":
"http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/1",
                "display": "Xellerate Users"
            }
        ]
    },
    "displayName": "Babs Jensen",
    "id": "145",
    "urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:User": {
        "createBy": {
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
        },
        "updateBy": {
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
        },
        "passwordExpireDate": "2015-05-06T03:24:16.000-07:00",
        "locked": {
            "duration": 0,
            "value": "0",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/0"
        }
    },
    "userName": "BJENSEN@EXAMPLE.COM",
    "emails": [
        {
            "value": "bjensen@example.com",
            "type": "work"
        }
    ],
    "active": true,
    "userType": "Contractor",
    "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User": {
        "employeeNumber": "701984",
        "manager": {
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1",
            "displayName": "new display"
        },
        "department": "Tour Operations",
        "organization": "Xellerate Users"
    },
    "preferredLanguage": "en-US",
    "phoneNumbers": [
        {
            "value": "555-555-4444",
            "type": "mobile"
        },
        {
            "value": "555-555-5555",
            "type": "work"
        }
    ],
    "name": {
        "middleName": "Jane",
        "familyName": "Jensen",
        "givenName": "Barbara",
        "honorificSuffix": "III"
    },
    "addresses": [
        {
            "region": "CA",
            "streetAddress": "100 Universal City Plaza",
            "formatted": "100 Universal City Plaza\nHollywood, CA 91608 USA",
            "postalCode": "91608",
            "locality": "Hollywood",
            "country": "USA",
            "type": "work"
        },
        {
            "formatted": "456 Hollywood Blvd\nHollywood, CA 91608 USA",
            "type": "home"
        }
    ],
    "groups": [
        {
            "value": "3",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Groups/3",
            "type": "direct"
        }
    ],
    "timezone": "America/Los_Angeles",
    "title": "Tour Guide",
    "meta": {
        "lastModified": "2015-01-06T03:24:17.000-08:00",
        "location": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/145",
        "created": "2015-01-06T03:24:17.000-08:00",
        "resourceType": "User"
    }
}
```

21.5.1.2 Modify User (PUT)

Request:

- **Operation and URI:** `PUT http://HOST_NAME:PORT/idaas/im/scim/v1/Users/355`
- **Header:**
  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`
- **Body:**

```
{
  "schemas":
  [
    "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:User",
    "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User",
    "urn:ietf:params:scim:schemas:core:2.0:User"
  ],
  "userName": "userName_user216_08_09.382323",
  "name":
  {
    "familyName": "familyName2_user216_08_09.382323"
  },
  "userType": "Contractor",
  "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:User":
  {
    "description": "description2_user216_08_09.382323"
  },
  "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:User":
  {
       "homeOrganization":
    {
      "value": "4",
      "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/4"
    }
  }
}
```

Response:

- **Status:** `HTTP/1.1 200 OK`
- **Body:**

```
{
```

```
            "schemas": [
                "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:User",
                "urn:ietf:params:scim:schemas:core:2.0:User",
                "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User",
                "urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:User"
            ],
            "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:User": {
                "passwordIsExpired": "0",
                "userLoginAttemptsCounter": 0,
                "ldapCommonNameGenerated": 0,
                "userPasswordResetAttemptsCounter": 0,
                "passwordWarnDate": "2015-07-02T08:46:57.000-07:00",
                "homeOrganization": {
                    "value": "1",
                    "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/1"
                },
                "passwordCreateDate": "2015-03-11T08:46:57.000-07:00",
                "provisionedDate": "2015-03-11T08:46:57.000-07:00",
                "passwordPolicyDescription": [
                    {
                        "value": "Password must not match or contain first name."
                    },
                    {
                        "value": "Password must not match or contain last name."
                    },
                    {
                        "value": "Password must contain at least 2 alphabetic
character(s)."
                    },
                    {
                        "value": "Password must be at least 6 character(s) long."
                    },
                    {
                        "value": "Password must contain at least 1 lowercase
letter(s)."
                    },
                    {
                        "value": "Password must contain at least 1 numeric
character(s)."
                    },
                    {
                        "value": "Password must contain at least 1 uppercase
letter(s)."
                    },
                    {
                        "value": "Password must start with an alphabetic character."
                    },
                    {
                        "value": "Password must not match or contain user ID."
                    }
                ],
                "userMustChangePasswordAtNextLogin": "1",
                "disabled": false,
                "organizations": [
                    {
                        "value": "1",
                        "$ref":
"http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/1",
                        "display": "Xellerate Users"
                    }
```

```
    ],
    "description": "description2_user216_08_09.382323"
},
"displayName": "Babs Jensen",
"id": "355",
"urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:User": {
    "createBy": {
        "value": "1",
        "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
    },
    "updateBy": {
        "value": "1",
        "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
    },
    "passwordExpireDate": "2015-07-09T08:46:57.000-07:00",
    "locked": {
        "duration": 0,
        "value": "0"
    }
},
"userName": "USERNAME_USER216_08_09.382323",
"emails": [
    {
        "value": "u1@example.com",
        "type": "work"
    }
],
"active": true,
"userType": "Contractor",
"urn:ietf:params:scim:schemas:extension:enterprise:2.0:User": {
    "employeeNumber": "701984",
    "manager": {
        "value": "1",
        "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1",
        "displayName": "display"
    },
    "department": "Tour Operations",
    "organization": "Xellerate Users"
},
"preferredLanguage": "en-US",
"phoneNumbers": [
    {
        "value": "555-555-4444",
        "type": "mobile"
    },
    {
        "value": "555-555-5555",
        "type": "work"
    }
],
"name": {
    "middleName": "Jane",
    "familyName": "familyName2_user216_08_09.382323",
    "givenName": "Barbara",
    "honorificSuffix": "III"
},
"addresses": [
    {
        "region": "CA",
        "streetAddress": "100 Universal City Plaza",
                "formatted": "100 Universal City Plaza\nHollywood, CA 91608 USA",
                "postalCode": "91608",
                "locality": "Hollywood",
                "country": "USA",
                "type": "work"
            },
            {
                "formatted": "456 Hollywood Blvd\nHollywood, CA 91608 USA",
                "type": "home"
            }
        ],
        "groups": [
            {
                "value": "3",
                "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Groups/3",
                "type": "direct"
            }
        ],
        "timezone": "America/Los_Angeles",
        "title": "Tour Guide",
        "meta": {
            "lastModified": "2015-03-11T08:47:19.000-07:00",
            "location": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/355",
            "created": "2015-03-11T08:46:57.000-07:00",
            "resourceType": "User"
        }
    }
```

21.5.1.3 Modify User (PATCH)

Request:

- **Operation and URI:** `PATCH http://HOST_NAME:PORT/idaas/im/scim/v1/Users/355`
- **Header:**
  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`
- **Body:**

```
{
  "schemas":
  [
    "urn:ietf:params:scim:api:messages:2.0:PatchOp"
  ],
  "Operations":
  [
    {
      "op":"replace",

"path":"urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:User:description,
      "value":"description3"
    }
  ]
}
```

Response:

- **Status:** `HTTP/1.1 200 OK`
- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:User",
        "urn:ietf:params:scim:schemas:core:2.0:User",
        "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User",
        "urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:User"
    ],
    "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:User": {
        "passwordIsExpired": "0",
        "userLoginAttemptsCounter": 0,
        "ldapCommonNameGenerated": 0,
        "userPasswordResetAttemptsCounter": 0,
        "passwordWarnDate": "2015-07-02T08:46:57.000-07:00",
        "homeOrganization": {
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/1"
        },
        "passwordCreateDate": "2015-03-11T08:46:57.000-07:00",
        "provisionedDate": "2015-03-11T08:46:57.000-07:00",
        "passwordPolicyDescription": [
            {
                "value": "Password must not match or contain first name."
            },
            {
                "value": "Password must not match or contain last name."
            },
            {
                "value": "Password must contain at least 2 alphabetic
character(s)."
            },
            {
                "value": "Password must be at least 6 character(s) long."
            },
            {
                "value": "Password must contain at least 1 lowercase
letter(s)."
            },
            {
                "value": "Password must contain at least 1 numeric
character(s)."
            },
            {
                "value": "Password must contain at least 1 uppercase
letter(s)."
            },
            {
                "value": "Password must start with an alphabetic character."
            },
            {
                "value": "Password must not match or contain user ID."
            }
        ],
        "userMustChangePasswordAtNextLogin": "1",
        "disabled": false,
        "organizations": [
            {
                "value": "1",
                "$ref":
"http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/1",
                "display": "Xellerate Users"
            }
        ],
        "description": "description3"
    },
    "displayName": "Babs Jensen",
    "id": "355",
    "urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:User": {
        "createBy": {
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
        },
        "updateBy": {
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
        },
        "passwordExpireDate": "2015-07-09T08:46:57.000-07:00",
        "locked": {
            "duration": 0,
            "value": "0"
        }
    },
    "userName": "USERNAME_USER216_08_09.382323",
    "emails": [
        {
            "value": "u1@example.com",
            "type": "work"
        }
    ],
    "active": true,
    "userType": "Contractor",
    "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User": {
        "employeeNumber": "701984",
        "manager": {
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1",
            "displayName": "display"
        },
        "department": "Tour Operations",
        "organization": "Xellerate Users"
    },
    "preferredLanguage": "en-US",
    "phoneNumbers": [
        {
            "value": "555-555-4444",
            "type": "mobile"
        },
        {
            "value": "555-555-5555",
            "type": "work"
        }
    ],
    "name": {
        "middleName": "Jane",
        "familyName": "familyName2_user216_08_09.382323",
        "givenName": "Barbara",
        "honorificSuffix": "III"
    },
    "addresses": [
        {
            "region": "CA",
            "streetAddress": "100 Universal City Plaza",
            "formatted": "100 Universal City Plaza\nHollywood, CA 91608 USA",
            "postalCode": "91608",
            "locality": "Hollywood",
            "country": "USA",
            "type": "work"
        },
        {
            "formatted": "456 Hollywood Blvd\nHollywood, CA 91608 USA",
            "type": "home"
        }
    ],
    "groups": [
        {
            "value": "3",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Groups/3",
            "type": "direct"
        }
    ],
    "timezone": "America/Los_Angeles",
    "title": "Tour Guide",
    "meta": {
        "lastModified": "2015-03-11T08:49:17.000-07:00",
        "location": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/355",
        "created": "2015-03-11T08:46:57.000-07:00",
        "resourceType": "User"
    }
}
```

21.5.1.4 View Users with Pagination

The following is an example of pagination:

Request:

- **Operation and URI:** `GET /Users`
  `http://HOST_NAME:PORT/idaas/im/scim/v1/Users?attributes=id&startIndex=6&count=5`

Response:

- **Status:** `HTTP/1.1 200 OK`
- **Body:**

```
    {
        "schemas": [
            "urn:ietf:params:scim:api:messages:2.0:ListResponse"
        ],
        "totalResults": 5,
        "itemsPerPage": 5,
        "startIndex": 6,
        "Resources": [
            {
                "id": "59"
            },
            {
                "id": "42"
            },
            {
                "id": "25"
            },
            {
                "id": "106"
            },
            {
                "id": "89"
            }
        ]
    }
```

The following are examples of search filters:

```
http://HOST_NAME:PORT/idaas/im/scim/v1/Users?filter=(userName co
xel)&attributes=id

http://HOST_NAME:PORT/idaas/im/scim/v1/Users?attributes=userName&filter=(userName
co 4) and (userName co BUG)

http://HOST_NAME:PORT/idaas/im/scim/v1/Users?attributes=userName&filter=(emails.ty
pe eq work and emails.value sw u)
```

Note: For a complete description of search filters, see the "Filtering" section of the SCIM REST API IETF draft at the following URL:

https://tools.ietf.org/html/draft-ietf-scim-api-14#section-3.2.2.2

21.5.1.5 Delete User

Request:

- **Operation and URI:** `DELETE http://HOST_NAME:PORT/idaas/im/scim/v1/Users/355`
- **Header:**
  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`
- **Body:** NA

Response:

- **Status:** `204 No Content`
- **Body:** NA

21.5.1.6 Lock User

Request:

- **Operation and URI:** `PATCH http://HOST_NAME:PORT/idaas/im/scim/v1/Users/356`
- **Header:**

  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`
- **Body:**

```
{
  "schemas": ["urn:ietf:params:scim:api:messages:2.0:PatchOp"],
  "Operations": [
  {
    "op":"replace",
    "path":"urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:User:locked",
    "value" :
      {
        "value" : 1,
        "duration" : 3600
      }
  }
  ]
}
```

Response:

Returns the full resource with lock attribute update.

Status: `HTTP/1.1 200 OK`

21.5.1.7 Unlock User

Request:

- **Operation and URI:** `PATCH http://`*`HOST_NAME`*`:`*`PORT`*`/idaas/im/scim/v1/Users/356`
- **Header:**
  - **Content-Type:** application/scim+json
  - **Authorization:** Bearer h480djs93hd8
- **Body:**

```
{
  "schemas": ["urn:ietf:params:scim:api:messages:2.0:PatchOp"],
  "Operations": [
  {
    "op":"replace",
    "path":"urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:User:locked",
    "value" :
      {
        "value" : 0
      }
  }
]
}
```

Response:

Returns the full resource with lock attribute update.

Status: `HTTP/1.1 200 OK`

21.5.1.8 Reset Password by Providing New Password

Request:

- **Operation and URI:** `PATCH http://HOST_NAME:PORT/idaas/im/scim/v1/Users/356`
- **Header:**
  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`
- **Body:**

```
{
      "schemas": ["urn:ietf:params:scim:api:messages:2.0:PatchOp"],
      "Operations": [
              {
                    "op": "replace","path":
"urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:User:passwd",
                    "value" :
                  {
                       "value": "newPassw0rd",
                       "sendNotification": "true",
                       "sendNotificationTo": "example2@example.com"
                  }

            }
      ]

}
```

Response:

Response contains the modified resource.

Status: `HTTP/1.1 200 OK`

21.5.1.9 Reset Password by Auto-Generated Password

Request:

- **Operation and URI:** `PATCH http://HOST_NAME:PORT/idaas/im/scim/v1/Users/356`
- **Header:**
  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`
- **Body:**

```
{
      "schemas": ["urn:ietf:params:scim:api:messages:2.0:PatchOp"],
      "Operations": [
              {
                    "op": "replace", "path":
"urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:User:passwd",
                    "value":
                  {
                       "value": "auto-generate",
                       "sendNotification": "true",
                       "sendNotificationTo": "john.doe@example.com"
                    }

                }
            ]

        }
```

Response

Response is the modified resource.

- **Status:** `HTTP/1.1 200 OK`

21.5.1.10 View User

Request:

- **Operation and URI:** `GET http://`*`HOST_NAME`*`:`*`PORT`*`/idaas/im/scim/v1/Me`
- **Header:**
  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`

Response:

- **Status:** `HTTP/1.1 200 OK`
- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:User",
        "urn:ietf:params:scim:schemas:core:2.0:User",
        "urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:User",
        "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User"
    ],
    "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:User": {
        "userLoginAttemptsCounter": 0,
        "ldapCommonNameGenerated": 0,
        "userPasswordResetAttemptsCounter": 0,
        "ldapCommonName": "System Administrator",
        "passwordWarnDate": "2015-06-30T01:51:27.000-07:00",
        "lastSuccessfulLoginDate": "2015-03-11T00:00:00.000-07:00",
        "homeOrganization": {
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/1"
        },
        "passwordPolicyDescription": [
            {
                "value": "Password must not match or contain first name."
            },
            {
                "value": "Password must not match or contain last name."
            },
            {
                "value": "Password must contain at least 2 alphabetic
character(s)."
            },
            {
                "value": "Password must be at least 6 character(s) long."
            },
            {
                "value": "Password must contain at least 1 lowercase
letter(s)."
            },
            {
                "value": "Password must contain at least 1 numeric
character(s)."
            },
            {
                "value": "Password must contain at least 1 uppercase
letter(s)."
            },
            {
                "value": "Password must start with an alphabetic character."
            },
            {
                "value": "Password must not match or contain user ID."
            }
        ],
        "disabled": false,
        "dataLevel": "2",
        "organizations": [
            {
                "value": "1",
                "$ref":
"http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/1",
                "display": "Xellerate Users"
            }
        ]
    },
    "displayName": "display",
    "id": "1",
    "urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:User": {
        "createBy": {
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
        },
        "updateBy": {
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
        },
        "passwordExpireDate": "2015-07-07T01:51:27.000-07:00",
        "locked": {
            "value": "0"
        }
    },
    "userName": "XELSYSADM",
    "emails": [
        {
            "value": "donotreply@example.com",
            "type": "work"
        }
    ],
    "active": true,
    "userType": "Full-Time",
    "name": {
        "familyName": "Administrator",
        "givenName": "System"
    },
    "groups": [
        {
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Groups/1",
            "type": "direct"
        },
        {
            "value": "6",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Groups/6",
            "type": "direct"
        }
    ],
    "meta": {
        "lastModified": "2015-03-11T08:15:44.000-07:00",
        "location": "http://HOST_NAME:PORT/idaas/im/scim/v1/Me",
        "created": "2015-03-09T01:51:27.000-07:00",
        "resourceType": "User"
    },
    "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User": {
        "organization": "Xellerate Users"
    }
}
```

21.5.1.11 Self Registration

Request:

- **Operation and URI:** `POST http://HOST_NAME:PORT/idaas/im/scim/v1/Me`
- **Header:**
  - **Content-Type:** `application/scim+json`
  - **Authorization:** NA
- **Body:**

```
{
  "schemas": ["urn:ietf:params:scim:schemas:core:2.0:User",
              "urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:User",
              "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:User",
              "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User"],
  "userName": "bjensen@example.com",
  "name": {
    "familyName": "Jensen",
    "givenName": "Barbara",
    "middleName": "Jane",
    "honorificSuffix": "III"
  },
  "displayName": "Babs Jensen",
  "emails": [
    {
      "value": "bjensen@example.com",
      "type": "work"
    }
  ],
  "userType": "Full-Time",
  "password":"t1meMa$heen",
  "urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:User": {
    "challenges": [
        {
            "challenge":"What is your favorite color?",
            "response":"color"
        },
        {
            "challenge":"What is the name of your pet?",
            "response":"pet"
        },
        {
            "challenge":"What is the city of your birth?",
            "response":"city"
        }
    ]
  }
}
```

Response:

User is created directly and entityId of the new user is returned.

- **Status:** `HTTP/1.1 201 Created`

21.5.1.12 Modify Self Profile (PATCH)

Request

- **Operation and URI:** `PATCH http://`*`HOST_NAME`*`:`*`PORT`*`/idaas/im/scim/v1/Me`
- **Header:**
  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`
- **Body:**

```
{
"schemas": ["urn:ietf:params:scim:api:messages:2.0:PatchOp"],
"Operations":[
  {
  "op":"replace",
  "path":"displayName",
  "value" : "NEW_NAME"
  }
]
}
```

Response:

- **Status:** `HTTP/1.1 200 OK`
- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:User",
        "urn:ietf:params:scim:schemas:core:2.0:User",
        "urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:User",
        "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User"
    ],
    "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:User": {
        "userLoginAttemptsCounter": 0,
        "ldapCommonNameGenerated": 0,
        "userPasswordResetAttemptsCounter": 0,
        "ldapCommonName": "System Administrator",
        "passwordWarnDate": "2015-06-30T01:51:27.000-07:00",
        "lastSuccessfulLoginDate": "2015-03-11T00:00:00.000-07:00",
        "homeOrganization": {
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/1"
        },
        "passwordPolicyDescription": [
            {
                "value": "Password must not match or contain first name."
            },
            {
                "value": "Password must not match or contain last name."
            },
            {
                "value": "Password must contain at least 2 alphabetic
character(s)."
            },
            {
                "value": "Password must be at least 6 character(s) long."
            },
            {
                "value": "Password must contain at least 1 lowercase
letter(s)."
            },
            {
                "value": "Password must contain at least 1 numeric
character(s)."
            },
            {
                "value": "Password must contain at least 1 uppercase
letter(s)."
            },
            {
                "value": "Password must start with an alphabetic character."
            },
            {
                "value": "Password must not match or contain user ID."
            }
        ],
        "disabled": false,
        "dataLevel": "2",
        "organizations": [
            {
                "value": "1",
                "$ref":
"http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/1",
                "display": "Xellerate Users"
            }
        ]
    },
    "displayName": "NEW_NAME",
    "id": "1",
    "urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:User": {
        "createBy": {
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
        },
        "updateBy": {
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
        },
        "passwordExpireDate": "2015-07-07T01:51:27.000-07:00",
        "locked": {
            "value": "0"
        }
    },
    "userName": "XELSYSADM",
    "emails": [
        {
            "value": "donotreply@example.com",
            "type": "work"
        }
    ],
    "active": true,
    "userType": "Full-Time",
    "name": {
        "familyName": "Administrator",
        "givenName": "System"
    },
    "groups": [
        {
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Groups/1",
            "type": "direct"
        },
        {
            "value": "6",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Groups/6",
            "type": "direct"
        }
    ],
    "meta": {
        "lastModified": "2015-03-11T08:55:23.000-07:00",
        "location": "http://HOST_NAME:PORT/idaas/im/scim/v1/Me",
        "created": "2015-03-09T01:51:27.000-07:00",
        "resourceType": "User"
    },
    "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User": {
        "organization": "Xellerate Users"
    }
}
```

21.5.1.13 Modify Profile (PUT)

Request:

- **Operation and URI:** `PUT http://HOST_NAME:PORT/idaas/im/scim/v1/Me`
- **Header:**
  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`
- **Body:**

```
{
  "schemas":
  [
    "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:User",
    "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User",
    "urn:ietf:params:scim:schemas:core:2.0:User"
  ],
  "userName": "bjensen@example.com",
  "name":
  {
    "familyName": "Jensen"
  },
  "userType": "Contractor",
  "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User":
  {
    "organization": "Xellerate Users",
    "homeOrganization":
    {
      "value": "1",
      "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/1"
    }
  }
}
```

Response:

- **Status:** `HTTP/1.1 200 OK`
- **Body:**

```
{
    "schemas":
    [
        "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:User",
        "urn:ietf:params:scim:schemas:core:2.0:User",
        "urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:User",
        "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User"
    ],
    "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:User": {
        "userLoginAttemptsCounter": 0,
        "passwordIsExpired": "0",
        "ldapCommonNameGenerated": 0,
        "userPasswordResetAttemptsCounter": 0,
        "passwordWarnDate": "2015-04-29T03:24:16.000-07:00",
        "homeOrganization": {
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/1"
        },
        "passwordCreateDate": "2015-01-06T03:24:16.000-08:00",
        "provisionedDate": "2015-01-06T03:24:16.000-08:00",
        "passwordPolicyDescription": [],
        "userMustChangePasswordAtNextLogin": "1",
        "disabled": false,
        "organizations": [
            {
                "value": "1",
                "$ref":
"http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/1",
                "display": "Xellerate Users"
            }
        ]
    },
    "displayName": "Babs Jensen",
    "id": "145",
    "urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:User": {
        "createBy": {
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
        },
        "updateBy": {
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
        },
        "passwordExpireDate": "2015-05-06T03:24:16.000-07:00",
        "locked": {
            "duration": 0,
            "value": "0",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/0"
        }
    },
    "userName": "BJENSEN@EXAMPLE.COM",
    "emails": [
        {
            "value": "bjensen@example.com",
            "type": "work"
        }
    ],
    "active": true,
    "userType": "Contractor",
    "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User": {
        "employeeNumber": "701984",
        "manager": {
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1",
            "displayName": "new display"
        },
        "department": "Tour Operations",
        "organization": "Xellerate Users"
    },
    "preferredLanguage": "en-US",
    "phoneNumbers": [
        {
            "value": "555-555-4444",
            "type": "mobile"
        },
        {
            "value": "555-555-5555",
            "type": "work"
        }
    ],
    "name": {
        "middleName": "Jane",
        "familyName": "Jensen",
        "givenName": "Barbara",
        "honorificSuffix": "III"
    },
    "addresses": [
        {
            "region": "CA",
            "streetAddress": "100 Universal City Plaza",
            "formatted": "100 Universal City Plaza\nHollywood, CA 91608 USA",
            "postalCode": "91608",
            "locality": "Hollywood",
            "country": "USA",
            "type": "work"
        },
        {
            "formatted": "456 Hollywood Blvd\nHollywood, CA 91608 USA",
            "type": "home"
        }
    ],
    "groups": [
        {
            "value": "3",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Groups/3",
            "type": "direct"
        }
    ],
    "timezone": "America/Los_Angeles",
    "title": "Tour Guide",
    "meta": {
        "lastModified": "2015-01-06T03:24:17.000-08:00",
        "location": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/145",
        "created": "2015-01-06T03:24:17.000-08:00",
        "resourceType": "User"
    }
}
```

21.5.1.14 PasswordResetterWithChallenges

Request:

- **Operation and URI:** `POST http://HOST_NAME:PORT/idaas/im/scim/v1/PasswordResetterWithChallenges`
- **Header:**
  - **Content-Type:** `application/json`
  - **Authorization:** NA
- **Body:**

```
{
  "schemas":

["urn:ietf:params:scim:schemas:oracle:core:2.0:IDM:PasswordResetterWithChalleng
es"],
  "userName": "JDOE",
  "challenges":
  [
        {
             "challenge":"What is the name of your pet?",
             "response":"name"
        },
        {
             "challenge":"What is the city of your birth?",
             "response":"city"
        },
        {
             "challenge":"What is your favorite color?",
             "response":"color"
        }
```

```
   ],
   "password": "Welcome3"
}
```

Response:

The response is empty.

- **Status:** `HTTP/1.1 204 No Content`

21.5.1.15 PasswordValidator

Request:

The following request is to validate a potential password. Successful response of validate password request means password is valid.

- **Operation and URI:** `POST http://`*`HOST_NAME`*`:`*`PORT`*`/idaas/im/scim/v1/PasswordValidator`
- **Header:**
  - **Content-Type:** `application/json`
  - **Authorization:** `Bearer h480djs93hd8`
- **Body:**

```
{
  "schemas":
  [
    "urn:ietf:params:scim:schemas:oracle:core:2.0:IDM:PasswordValidator"
  ],
  "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1",
  "password": "jijijSSij1"
}
```

Response:

- **Status:** `HTTP/1.1 204 No Content`

21.5.1.16 UserNameValidator

Request:

Successful response of validate user name request means password is valid. The request is as follows:

- **Operation and URI:** `POST http://`*`HOST_NAME`*`:`*`PORT`*`/idaas/im/scim/v1/UserNameValidator`
- **Header:**
  - **Content-Type:** `application/json`
  - **Authorization:** `Bearer h480djs93hd8`
- **Body:**

```
{

"schemas":["urn:ietf:params:scim:schemas:oracle:core:2.0:IDM:UserNameValidator"
],
  "userName": "aUserName"
}
```

Response:

- **Status:** `HTTP/1.1 204 No Content`

21.5.1.17 UserNameGenerator

Request:

- **Operation and URI:** `POST http://`*`HOST_NAME`*`:`*`PORT`*`/idaas/im/scim/v1/UserNameGenerator`
- **Header:**
  - **Content-Type:** `application/json`
  - **Authorization:** `Bearer h480djs93hd8`
- **Body:**

```
{

"schemas":["urn:ietf:params:scim:schemas:oracle:core:2.0:IDM:UserNameGenerator"
],
  "name":
  {
    "formatted": "Ms. Barbara J Doe III",
    "familyName": "Doe",
    "givenName": "Barbara",
    "middleName": "Jane",
    "honorificSuffix": "III"
  }
}
```

Response:

- **Status:** `HTTP/1.1 201 Created`
- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:schemas:core:2.0:User",
        "urn:ietf:params:scim:schemas:oracle:core:2.0:IDM:UserNameGenerator"
    ],
    "meta": {
        "location": "http://HOST_NAME:PORT/idaas/im/scim/v1/UserNameGenerator",
        "resourceType": "UserNameGenerator"
    },
    "urn:ietf:params:scim:schemas:core:2.0:User": {
        "userName": "Barbara.Doe@example.com"
    }
}
```

21.5.1.18 UserNameRecoverer

Request:

The request is as follows (no authorization header, unauthenticated flow):

- **Operation and URI:** `POST http://HOST_NAME:PORT/idaas/im/scim/v1/UserNameRecoverer`
- **Header:**
  - **Content-Type:** `application/json`
  - **Authorization:** NA
- **Body:**

```
{
  "schemas":
["urn:ietf:params:scim:schemas:oracle:core:2.0:IDM:UserNameRecoverer"],
  "email": "myemail@example.com"
}
```

Response:

- **Status:** `HTTP/1.1 204 No Content`

The username is sent to the user's email address.

21.5.2 Role Management

This section provides the following examples of the group resource:

- View Role
- Create Role
- Modify Role (PUT)
- Modify Role (PATCH)
- Delete Role
- Remove Role (PATCH)

21.5.2.1 View Role

Request:

- **Operation and URI:** `GET http://HOST_NAME:PORT/idaas/im/scim/v1/Groups?attributes=id,displayName`

Response:

- **Status:** `HTTP/1.1 200 OK`
- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:api:messages:2.0:ListResponse"
    ],
    "totalResults": 4,
    "Resources": [
        {
            "displayName": "Group1",
            "id": "2"
        },
        {
            "displayName": "SYSTEM ADMINISTRATORS",
            "id": "3"
        },
        {
            "displayName": "Group2",
            "id": "4"
        },
        {
            "displayName": "Group3",
            "id": "5"
        }
    ]
}
```

21.5.2.2 Create Role

Request:

- **Operation and URI:** `POST http://HOST_NAME:PORT/idaas/im/scim/v1/Groups`
- **Header:**
  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`
- **Body:**

```
{
    "schemas":
    [
        "urn:ietf:params:scim:schemas:core:2.0:Group",
        "urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:Group",
        "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:Group"
    ],
    "displayName": "Group33",
    "urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:Group":
    {
        "email": "group33@example.com",
        "description": "description1"
    },
    "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:Group":
    {
        "namespace": "Default"
    }
}
```

Response:

- **Status:** `HTTP/1.1 201 Created`
- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:Group",
        "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:Group",
        "urn:ietf:params:scim:schemas:core:2.0:Group"
    ],
    "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:Group": {
        "organizationsPublishedTo": [
            {
                "value": "3",
                "$ref":
"http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/3"
            }
        ],
        "category": {
            "name": "Default",
            "value": 1
        },
        "namespace": "Default",
        "catalog": {
            "tags": "Group33 Group33 Default",
            "requestable": true,
            "certifiable": false,
            "id": "151",
            "categoryName": "Role",
            "auditable": false,
            "itemRisk": 3,
            "hierarchicalDataAvailable": false
        }
    },
    "urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:Group": {
        "createBy": {
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
        },
        "description": "description1",
        "updateBy": {
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
        },
        "owner": {
            "lastName": "Administrator",
            "email": "donotreply@example.com",
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1",
            "login": "XELSYSADM",
            "firstName": "System",
            "displayName": "NEW_NAME"
        },
        "email": "group33@example.com"
    },
    "meta": {
        "lastModified": "2015-03-11T08:55:57.000-07:00",
        "location": "http://HOST_NAME:PORT/idaas/im/scim/v1/Groups/157",
        "created": "2015-03-11T08:55:57.000-07:00",
        "resourceType": "Group"
    },
    "displayName": "Group33",
    "id": "157"
}
```

21.5.2.3 Modify Role (PUT)

Request:

- **Operation and URI:** `PUT http://`*`HOST_NAME`*`:`*`PORT`*`/idaas/im/scim/v1/Groups/157`
- **Header:**
  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`
- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:Group",
        "urn:ietf:params:scim:schemas:core:2.0:Group",
        "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:Group"
    ],
    "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:Group": {
        "organizationsPublishedTo": [
            {
                "value": "3",
                "$ref":
"http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/3"
            }
        ],
        "category": {
            "name": "Default",
            "value": 1
        },
        "namespace": "Default"
    },
    "urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:Group": {
        "description": "description1",
        "localeNames": [
            {
                "name": "Group_group09_53_11.228163",
                "locale": "base"
            }
        ],

        "email": "group_new@example.com"
    },
    "displayName": "Group_group09_53_11.228163"
}
```

Response:

- **Status:** `HTTP/1.1 200 OK`
- **Location:** `https://`*`HOST_NAME`*`:`*`PORT`*`/Groups/157`
- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:Group",
        "urn:ietf:params:scim:schemas:core:2.0:Group",
        "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:Group"
    ],
    "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:Group": {
        "organizationsPublishedTo": [
            {
                "value": "3",
                "$ref": 
"http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/3"
            }
        ],
        "category": {
            "name": "Default",
            "value": 1
        },
        "namespace": "Default"
    },
    "urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:Group": {
        "createBy": {
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
        },
        "description": "description1",
        "localeNames": [
            {
                "name": "Group_group09_53_11.228163",
                "locale": "base"
            }
        ],
        "updateBy": {
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
        },
        "owner": {
            "lastName": "Administrator",
            "email": "donotreply@example.com",
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1",
            "login": "XELSYSADM",
            "displayName": "System Administrator",
            "firstName": "System"
        },
        "email": "group_new@example.com"
    },
    "meta": {
        "lastModified": "2015-01-05T06:59:25.000-08:00",
        "location": "http://HOST_NAME:PORT/idaas/im/scim/v1/Groups/157",
        "created": "2015-01-05T06:55:14.000-08:00",
        "resourceType": "Group"
    },
    "displayName": "Group_group09_53_11.228163",
    "id": "157"
}
```

21.5.2.4 Modify Role (PATCH)

Request:

- **Operation and URI:** `PATCH http://HOST_NAME:PORT/idaas/im/scim/v1/Groups/153`
- **Header:**
  - **Content-Type:** `application/scim+json`

- **Authorization:** `Bearer h480djs93hd8`

- **Body:**

```
{
  "schemas":
  [
    "urn:ietf:params:scim:api:messages:2.0:PatchOp"
  ],
  "Operations":
  [
    {
      "op":"replace"          ,

"path":"urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:Group:description
",
      "value":"description3"
    }
  ]
}
```

Response:

- **Status:** `HTTP/1.1 200 OK`
- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:Group",
        "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:Group",
        "urn:ietf:params:scim:schemas:core:2.0:Group"
    ],
    "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:Group": {
        "organizationsPublishedTo": [
            {
                "value": "3",
                "$ref":
"http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/3"
            }
        ],
        "category": {
            "name": "Default",
            "value": 1
        },
        "namespace": "Default",
        "catalog": {
            "tags": "replace_catalog_requestable_6587843
replace_catalog_requestable_6587843 Default",
            "requestable": true,
            "certifiable": false,
            "id": "147",
            "categoryName": "Role",
            "auditable": false,
            "itemRisk": 3,
            "hierarchicalDataAvailable": false
        }
    },
    "urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:Group": {
        "createBy": {
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
        },
        "description": "description3",
        "updateBy": {
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
        },
        "owner": {
            "lastName": "Administrator",
            "email": "donotreply@example.com",
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1",
            "login": "XELSYSADM",
            "firstName": "System",
            "displayName": "NEW_NAME"
        }
    },
    "meta": {
        "lastModified": "2015-03-11T08:59:16.000-07:00",
        "location": "http://HOST_NAME:PORT/idaas/im/scim/v1/Groups/153",
        "created": "2015-03-11T08:13:11.000-07:00",
        "resourceType": "Group"
    },
    "displayName": "replace_catalog_requestable_6587843",
    "id": "153"
}
```

21.5.2.5 Delete Role

Request:

- **Operation and URI:** `DELETE http://HOST_NAME:PORT/idaas/im/scim/v1/Groups/153`
- **Header:**
  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`

Response:

- **Status:** `HTTP/1.1 204 No Content`

21.5.2.6 Remove Role (PATCH)

This section provides an example of the Remove Role operation using the PATCH operation type. It contains the following topics:

Request

- **Operation and URI:** `PATCH http://HOST_NAME:PORT/iam/governance/scim/v1/Groups/<role_id>`
- **Header:**
  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`
- **Body:**

```
{
 "schemas":
 [
   "urn:ietf:params:scim:api:messages:2.0:PatchOp"
 ],
 "Operations":
 [
   {
     "op":"remove",
     "path":"urn:ietf:params:scim:schemas:core:2.0:Group:members",
     "value":[
      {
      "value":"<usr_key>",
      "$ref":"http://HOST_NAME:PORT/idaas/im/scim/v1/Users/<usr_key>"
      }
      ]
   }
 ]
}
```

Response:

- **Status:** `HTTP/1.1 200 OK`
- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:Group",
        "urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:Group",
        "urn:ietf:params:scim:schemas:core:2.0:Group"
    ],
    "urn:ietf:params:scim:schemas:extension:oracle:2.0:OIG:Group": {
        "namespace": "Default",
        "organizationsPublishedTo": [
            {
                "value": "3",
                "$ref":
"http://HOST_NAME:PORT/iam/governance/scim/v1/Organizations/3"
            }
        ],
        "category": {
            "name": "Default",
            "value": 1
        },
        "ldapDn": "cn=tesrole1,cn=groups,dc=isc,dc=com",
        "catalog": {
            "auditable": true,
            "hierarchicalDataAvailable": false,
            "id": "101",
            "requestable": true,
            "itemRisk": 3,
            "certifiable": true,
            "categoryName": "Role",
            "tags": "tesrole1 tesrole1 Default"
        }
    },
    "urn:ietf:params:scim:schemas:extension:oracle:2.0:IDM:Group": {
        "createBy": {
            "$ref": "http://HOST_NAME:PORT/iam/governance/scim/v1/Users/1",
            "value": "1"
        },
        "owner": {
            "firstName": "System",
            "lastName": "Administrator",
            "displayName": "System Administrator",
            "login": "XELSYSADM",
            "value": "1",
            "$ref": "http://HOST_NAME:PORT/iam/governance/scim/v1/Users/1",
            "email": "donotreply@oracle.com"
        },
        "updateBy": {
            "value": "5",
            "$ref": "http://HOST_NAME:PORT/iam/governance/scim/v1/Users/5"
        }
    },
    "members": [
        {
            "value": "7002",
            "$ref": "http://HOST_NAME:PORT/iam/governance/scim/v1/Users/7002"
        }
    ],
    "meta": {
        "created": "2018-11-13T08:48:49.000+05:30",
        "location":
"http://HOST_NAME:PORT/iam/governance/scim/v1/Groups/<role_id>",
        "lastModified": "2018-11-13T08:53:58.000+05:30",
        "resourceType": "Group"
    },
    "displayName": "<role name>",
    "id": "<role_d>"
}
```

21.5.3 Organization Management

This section provides the following examples of the organization resource:

- View Organization
- Create Organization
- Modify Organization (PUT)
- Modify Organizations (PATCH)
- Delete Organization

21.5.3.1 View Organization

Request:

- **Operation and URI:** `GET http://`*`HOST_NAME`*`:`*`PORT`*`/idaas/im/scim/v1/Organizations/148`
- **Header:**
  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`

Response:

- **Status:** `HTTP/1.1 200 OK`

- **Location:** `https://HOST_NAME:PORT/Organization/148`
- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:schemas:oracle:core:2.0:OIG:Organization"
    ],
    "parent": {
        "name": "Top",
        "value": "3",
        "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/3"
    },
    "name": "org_pcu3_1426086587854",
    "passwordPolicy": {
        "name": "ppchg_1426086587854",
        "value": "94"
    },
    "id": "148",
    "meta": {
        "lastModified": "2015-03-11T08:16:00.000-07:00",
        "location": "http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/148",
        "created": "2015-03-11T08:16:00.000-07:00",
        "resourceType": "Organization"
    },
    "customerType": "Company",
    "createBy": {
        "value": "1",
        "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
    },
    "enforceNewPasswordPolicy": "Yes",
    "updateBy": {
        "value": "1",
        "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
    },
    "status": "Active",
    "members": [
        {
            "value": "353",
            "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/353"
        }
    ]
}
```

21.5.3.2 Create Organization

Request:

- **Operation and URI:** `POST http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations`
- **Header:**
  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`
- **Body:**

```
{
  "schemas":
  [
    "urn:ietf:params:scim:schemas:oracle:core:2.0:OIG:Organization"
  ],
  "urn:ietf:params:scim:schemas:oracle:core:2.0:OIG:Organization":
  {
    "name": "organization16_08_50.141529",
    "customerType": "Branch"
  }
}
```

Note: The valid values for the `customerType` attribute are `Branch`, `Company`, and `Department`.

Response:

- **Status:** `HTTP/1.1 201 Created`
- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:schemas:oracle:core:2.0:OIG:Organization"
    ],
    "parent": {
        "name": "Top",
        "value": "3",
        "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/3"
    },
    "name": "organization16_08_50.141529",
    "id": "77",
    "meta": {
        "lastModified": "2015-02-06T07:06:46.000-08:00",
        "location": "http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/77",
        "created": "2015-02-06T07:06:46.000-08:00",
        "resourceType": "Organization"
    },
    "customerType": "Branch",
    "createBy": {
        "value": "1",
        "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
    },
    "enforceNewPasswordPolicy": "Yes",
    "updateBy": {
        "value": "1",
        "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
    },
    "status": "Active"
}
```

21.5.3.3 Modify Organization (PUT)

Request:

- **Operation and URI:** `PUT http://`*`HOST_NAME`*`:`*`PORT`*`/idaas/im/scim/v1/Organizations/77`
- **Header:**
  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`

- **Body:**

```
{
  "schemas":
  [
    "urn:ietf:params:scim:schemas:oracle:core:2.0:OIG:Organization"
  ],
  "urn:ietf:params:scim:schemas:oracle:core:2.0:OIG:Organization":
  {
    "name": "organization16_08_53.883452",
    "customerType": "Scim2"
  }
}
```

Response:

- **Status:** `HTTP/1.1 200 OK`
- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:schemas:oracle:core:2.0:OIG:Organization"
    ],
    "parent": {
        "name": "Top",
        "value": "3",
        "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/3"
    },
    "name": "organization16_08_53.883452",
    "id": "77",
    "meta": {
        "lastModified": "2015-02-06T07:09:27.000-08:00",
        "location": "http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/77",
        "created": "2015-02-06T07:06:46.000-08:00",
        "resourceType": "Organization"
    },
    "customerType": "Scim2",
    "createBy": {
        "value": "1",
        "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
    },
    "enforceNewPasswordPolicy": "Yes",
    "updateBy": {
        "value": "1",
        "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
    },
    "status": "Active"
}
```

21.5.3.4 Modify Organizations (PATCH)

Request:

- **Operation and URI:** `PATCH http://`*`HOST_NAME`*`:`*`PORT`*`/idaas/im/scim/v1/Organizations/77`
- **Header:**

  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`

- **Body:**

```
{
  "schemas":
  [
    "urn:ietf:params:scim:api:messages:2.0:PatchOp"
  ],
  "Operations":
  [
    {
      "op":"replace",
      "path":"customerType",
      "value":"Scim3"
    }
  ]
}
```

Response:

- **Status:** `HTTP/1.1 200 OK`
- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:schemas:oracle:core:2.0:OIG:Organization"
    ],
    "parent": {
        "name": "Top",
        "value": "3",
        "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/3"
    },
    "name": "organization16_08_53.883452",
    "id": "77",
    "meta": {
        "lastModified": "2015-02-06T07:13:19.000-08:00",
        "location": "http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/77",
        "created": "2015-02-06T07:06:46.000-08:00",
        "resourceType": "Organization"
    },
    "customerType": "Scim3",
    "createBy": {
        "value": "1",
        "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
    },
    "enforceNewPasswordPolicy": "Yes",
    "updateBy": {
        "value": "1",
        "$ref": "http://HOST_NAME:PORT/idaas/im/scim/v1/Users/1"
    },
    "status": "Active"
}
```

21.5.3.5 Delete Organization

Request:

- **Operation and URI:** `DELETE http://HOST_NAME:PORT/idaas/im/scim/v1/Organizations/77`
- **Header:**
  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`

Response:

- **Status:** `HTTP/1.1 204 No Content`

21.5.4 Password Policy Management

This section provides the following examples of the password policy resource:

- View Password Policy
- Create Password Policy
- Modify Password Policy (PUT)
- Modify Password Policy (PATCH)
- Delete Password Policy

21.5.4.1 View Password Policy

Request:

- **Operation and URI:** `GET http://HOST_NAME:PORT/idaas/im/scim/v1/PasswordPolicies?filter=(urn:ietf:params:scim:schemas:oracle:core:2.0:IDM:PasswordPolicy:allowedChars co q)&attributes=id,description,name`

Response:

- **Status:** `HTTP/1.1 200 OK`
- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:api:messages:2.0:ListResponse"
    ],
    "totalResults": 3,
    "Resources": [
        {
            "id": "2",
            "name": "pwp1_1423213466123",
            "description": "1"
        },
        {
            "id": "6",
            "name": "pwp_q_3466186",
            "description": "pwp_q_3466186"
        },
        {
            "id": "36",
            "name": "p1214_1423213645161"
        }
    ]
}
```

21.5.4.2 Create Password Policy

Request:

- **Operation and URI:** `POST http://HOST_NAME:PORT/idaas/im/scim/v1/PasswordPolicies`
- **Header:**
  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`
- **Body:**

```
{
  "schemas":
  [
    "urn:ietf:params:scim:schemas:oracle:core:2.0:IDM:PasswordPolicy"
  ],
  "urn:ietf:params:scim:schemas:oracle:core:2.0:IDM:PasswordPolicy":
  {
    "name":"pwdpol0901258070",
    "description":"complex password policy for ST org",
    "passwordWarningAfterInDays": 20,
    "passwordExpiresAfterInDays": 30,
    "minPasswordAgeInDays": 10,
    "userIdDisallowed": "false",
    "minLength": 3,
    "maxLength": 8,
    "firstNameDisallowed": "true",
    "challengeSource": 1,
    "challengeMinQuestions": 3,
    "challengeMinAnswers": 2,
    "challengeResponseMinLength": 5,
    "challengeAllowDuplicateResponses": "false",
    "challengeMaxIncorrectAttempts": 5,
    "challengeDefaultQuestions":
    [
      {
        "value": "what is your favorite color"
      },
      {
        "value": "what is name of your pet"
      },
      {
        "value": "which is your favorite movie"
      },
      {
        "value": "which is your favorite sport"
      }
    ]
  }
}
```

Response:

- **Status:** `HTTP/1.1 201 Created`
- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:schemas:oracle:core:2.0:IDM:PasswordPolicy"
    ],
    "challengesEnabled": false,
    "id": "49",
    "challengeSource": 1,
    "minLength": 3,
    "startsWithAlphabet": false,
    "description": "complex password policy for ST org",
    "name": "pwdpol0901258070",
    "complexPolicy": false,
    "challengeDefaultQuestions": [
        {
            "value": "what is your favorite color"
        },
        {
            "value": "what is name of your pet"
        },
        {
            "value": "which is your favorite movie"
        },
        {
            "value": "which is your favorite sport"
        }
    ],
    "challengeAllAtOnce": true,
    "minPasswordAgeInDays": 10,
    "passwordWarningAfterInDays": 20,
    "challengeResponseMinLength": 5,
    "userIdDisallowed": false,
    "maxLength": 8,
    "challengeMinQuestions": 3,
    "meta": {
        "location":
"http://HOST_NAME:PORT/idaas/im/scim/v1/PasswordPolicies/49",
        "resourceType": "PasswordPolicy"
    },
    "challengeMaxIncorrectAttempts": 5,
    "challengeMinAnswers": 2,
    "passwordExpiresAfterInDays": 30,
    "challengeAllowDuplicateResponses": false,
    "lastNameDisallowed": false,
    "firstNameDisallowed": true,
    "dictionaryDelimiter": "\u0000"
}
```

21.5.4.3 Modify Password Policy (PUT)

Request:

- **Operation and URI:** `PUT http://`*`HOST_NAME`*`:`*`PORT`*`/idaas/im/scim/v1/PasswordPolicies/49`
- **Header:**

- **Content-Type:** `application/scim+json`
- **Authorization:** `Bearer h480djs93hd8`

- **Body:**

```
{
  "schemas":
  [
    "urn:ietf:params:scim:schemas:oracle:core:2.0:IDM:PasswordPolicy"
  ],
  "urn:ietf:params:scim:schemas:oracle:core:2.0:IDM:PasswordPolicy":
  {
    "name":"pwdpol0905770857",
    "description":"complex password policy for ST org",
    "passwordWarningAfterInDays": 20,
    "passwordExpiresAfterInDays": 30,
    "minPasswordAgeInDays": 10,
    "userIdDisallowed": "false",
    "minLength": 3,
    "maxLength": 8,
    "firstNameDisallowed": "true",
    "challengeSource": 1,
    "challengeMinQuestions": 3,
    "challengeMinAnswers": 2,
    "challengeResponseMinLength": 5,
    "challengeAllowDuplicateResponses": "false",
    "challengeMaxIncorrectAttempts": 5,
    "challengeDefaultQuestions":
    [
      {
        "value": "what is your favorite car"
      },
      {
        "value": "what is name of your truck"
      },
      {
        "value": "which is your favorite bicycle"
      },
      {
        "value": "which is your favorite shoe"
      }
    ]
  }
}
```

Response:

- **Status:** `HTTP/1.1 200 OK`
- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:schemas:oracle:core:2.0:IDM:PasswordPolicy"
    ],
    "challengesEnabled": false,
    "id": "49",
    "challengeSource": 1,
    "minLength": 3,
    "startsWithAlphabet": false,
    "description": "complex password policy for ST org",
    "name": "pwdpol0905770857",
    "complexPolicy": false,
    "challengeDefaultQuestions": [
        {
            "value": "what is your favorite car"
        },
        {
            "value": "what is name of your truck"
        },
        {
            "value": "which is your favorite bicycle"
        },
        {
            "value": "which is your favorite shoe"
        }
    ],
    "challengeAllAtOnce": true,
    "minPasswordAgeInDays": 10,
    "passwordWarningAfterInDays": 20,
    "challengeResponseMinLength": 5,
    "userIdDisallowed": false,
    "maxLength": 8,
    "challengeMinQuestions": 3,
    "meta": {
        "location":
"http://HOST_NAME:PORT/idaas/im/scim/v1/PasswordPolicies/49",
        "resourceType": "PasswordPolicy"
    },
    "challengeMaxIncorrectAttempts": 5,
    "challengeMinAnswers": 2,
    "passwordExpiresAfterInDays": 30,
    "challengeAllowDuplicateResponses": false,
    "lastNameDisallowed": false,
    "firstNameDisallowed": true,
    "dictionaryDelimiter": "\u0000"
}
```

21.5.4.4 Modify Password Policy (PATCH)

Request:

- **Operation and URI:** `PATCH http://`*`HOST_NAME`*`:`*`PORT`*`/idaas/im/scim/v1/PasswordPolicies/49`
- **Header:**
  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`
- **Body:**

```
{
  "schemas":
  [
    "urn:ietf:params:scim:api:messages:2.0:PatchOp"
  ],
  "Operations":
  [
    {
      "op":"replace",
      "path":"firstNameDisallowed",
      "value":"false"
    }
  ]
}
```

Response:

- **Status:** `HTTP/1.1 200 OK`
- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:schemas:oracle:core:2.0:IDM:PasswordPolicy"
    ],
    "challengesEnabled": false,
    "id": "49",
    "challengeSource": 1,
    "minLength": 3,
    "startsWithAlphabet": false,
    "description": "complex password policy for ST org",
    "name": "pwdpol0905770857",
    "complexPolicy": false,
    "challengeDefaultQuestions": [
        {
            "value": "what is your favorite car"
        },
        {
            "value": "what is name of your truck"
        },
        {
            "value": "which is your favorite bicycle"
        },
        {
            "value": "which is your favorite shoe"
        }
    ],
    "challengeAllAtOnce": true,
    "minPasswordAgeInDays": 10,
    "passwordWarningAfterInDays": 20,
    "challengeResponseMinLength": 5,
    "userIdDisallowed": false,
    "maxLength": 8,
    "challengeMinQuestions": 3,
    "meta": {
        "location":
"http://HOST_NAME:PORT/idaas/im/scim/v1/PasswordPolicies/49",
        "resourceType": "PasswordPolicy"
    },
    "challengeMaxIncorrectAttempts": 5,
    "challengeMinAnswers": 2,
    "passwordExpiresAfterInDays": 30,
    "challengeAllowDuplicateResponses": false,
    "lastNameDisallowed": false,
    "firstNameDisallowed": false,
    "dictionaryDelimiter": "\u0000"
}
```

21.5.4.5 Delete Password Policy

Request:

- **Operation and URI:** `DELETE http://HOST_NAME:PORT/idaas/im/scim/v1/PasswordPolicies/49`
- **Header:**
  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`

Response:

- **Status:** `HTTP/1.1 204 NO Content`

21.5.5 Notification Template Management

This section provides the following examples of the notification template resource:

- View Notification Template
- Create Notification Template
- Modify Notification Template (PUT)
- Modify Notification Template (PATCH)
- Notification Template Management: Delete

21.5.5.1 View Notification Template

Request:

- **Operation and URI:** `GET http://HOST_NAME:PORT/idaas/im/scim/v1/NotificationTemplates?attributes =id,name`

Response:

- **Status:** `HTTP/1.1 200 OK`
- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:api:messages:2.0:ListResponse"
    ],
    "totalResults": 4,
    "Resources": [
        {
            "name": "AddProxyNotificationTemplate",
            "id": "12"
        },
        {
            "name": "BulkRequestCreation",
            "id": "6"
        },
        {
            "name": "CreateUserSelfServiceNotification",
            "id": "7"
        },
        {
            "name": "UserDeletedNotificationTemplate",
            "id": "10"
        }
    ]
}
```

21.5.5.2 Create Notification Template

Request:

- **Operation and URI:** `POST http://HOST_NAME:PORT/idaas/im/scim/v1/NotificationTemplates/`
- **Header:**
  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`
- **Body:**

```
{
  "schemas":
  [
   "urn:ietf:params:scim:schemas:oracle:core:2.0:OIG:NotificationTemplate"
  ],
  "urn:ietf:params:scim:schemas:oracle:core:2.0:OIG:NotificationTemplate":
  {
    "name":"notificationtemplate16_09_14.724166_1",
    "eventName":"AddProxy",
    "description":"myTemplate",
    "locales":
    [
      {
        "locale": "dddd",
        "subject": "Notification for contractors 1",
        "encoding": "UTF-8",
        "contentType": "text/html, charset=UTF-8",
        "shortMessage": "short message1",
        "longMessage": "long message1"
      },
      {
        "locale": "eeee",
        "subject": "Notification for contractors 2",
        "encoding": "UTF-8",
        "contentType": "text/html, charset=UTF-8",
        "shortMessage": "short message2",
        "longMessage": "long message2"
      }
    ]
  }
}
```

Response:

- **Status:** `HTTP/1.1 201 Created`
- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:schemas:oracle:core:2.0:OIG:NotificationTemplate"
    ],
    "meta": {
        "location":
"http://HOST_NAME:PORT/idaas/im/scim/v1/NotificationTemplates/25",
        "resourceType": "NotificationTemplate"
    },
    "name": "notificationtemplate16_09_14.724166_1",
    "eventName": "AddProxy",
    "locales": [
        {
            "subject": "Notification for contractors 2",
            "locale": "eeee",
            "shortMessage": "short message2",
            "encoding": "UTF-8",
            "contentType": "text/html, charset=UTF-8",
            "longMessage": "long message2"
        },
        {
            "subject": "Notification for contractors 1",
            "locale": "dddd",
            "shortMessage": "short message1",
            "encoding": "UTF-8",
            "contentType": "text/html, charset=UTF-8",
            "longMessage": "long message1"
        }
    ],
    "description": "myTemplate",
    "id": "25"
}
```

21.5.5.3 Modify Notification Template (PUT)

Request:

- **Operation and URI:** `PUT http://HOST_NAME:PORT/idaas/im/scim/v1/NotificationTemplates/25`
- **Header:**
  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`
- **Body:**

```
{
  "schemas":
  [
    "urn:ietf:params:scim:schemas:oracle:core:2.0:OIG:NotificationTemplate"
  ],
  "urn:ietf:params:scim:schemas:oracle:core:2.0:OIG:NotificationTemplate":
  {
    "name":"notificationtemplate16_09_17.742004_2",
    "eventName":"AddProxy",
    "description":"Template to notify contractors after password has been
reset",
    "locales":
    [
      {
        "subject": "Notification for contractors 1",
        "encoding": "UTF-8",
        "contentType": "text/html, charset=UTF-8",
        "shortMessage": "short message1",
        "longMessage": "long message1",
        "locale": "ffff"
      },
      {
        "subject": "Notification for contractors 2",
        "encoding": "UTF-8",
        "contentType": "text/html, charset=UTF-8",
        "shortMessage": "short message2",
        "longMessage": "long message2",
        "locale": "gggg"
      }
    ]
  }
}
```

Response:

- **Status:** `HTTP/1.1 200 OK`
- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:schemas:oracle:core:2.0:OIG:NotificationTemplate"
    ],
    "meta": {
        "location":
"http://HOST_NAME:PORT/idaas/im/scim/v1/NotificationTemplates/25",
        "resourceType": "NotificationTemplate"
    },
    "name": "notificationtemplate16_09_14.724166_1",
    "eventName": "AddProxy",
    "locales": [
        {
            "subject": "Notification for contractors 2",
            "locale": "gggg",
            "shortMessage": "short message2",
            "encoding": "UTF-8",
            "contentType": "text/html, charset=UTF-8",
            "longMessage": "long message2"
        },
        {
            "subject": "Notification for contractors 1",
            "locale": "ffff",
            "shortMessage": "short message1",
            "encoding": "UTF-8",
            "contentType": "text/html, charset=UTF-8",
            "longMessage": "long message1"
        }
    ],
    "description": "Template to notify contractors after password has been
reset",
    "id": "25"
}
```

21.5.5.4 Modify Notification Template (PATCH)

Request:

- **Operation and URI:** `PATCH http://HOST_NAME:PORT/idaas/im/scim/v1/NotificationTemplates/25`
- **Header:**
  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`
- **Body:**

```
{
  "schemas":
  [
    "urn:ietf:params:scim:api:messages:2.0:PatchOp"
  ],
  "Operations":
  [
    {
      "op":"replace",
      "path":"description",
      "value":"description3"
    }
  ]
}
```

Response:

- **Status:** `HTTP/1.1 200 OK`
- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:schemas:oracle:core:2.0:OIG:NotificationTemplate"
    ],
    "meta": {
        "location":
"http://HOST_NAME:PORT/idaas/im/scim/v1/NotificationTemplates/25",
        "resourceType": "NotificationTemplate"
    },
    "name": "notificationtemplate16_09_14.724166_1",
    "eventName": "AddProxy",
    "locales": [
        {
            "subject": "Notification for contractors 2",
            "locale": "gggg",
            "shortMessage": "short message2",
            "encoding": "UTF-8",
            "contentType": "text/html, charset=UTF-8",
            "longMessage": "long message2"
        },
        {
            "subject": "Notification for contractors 1",
            "locale": "ffff",
            "shortMessage": "short message1",
            "encoding": "UTF-8",
            "contentType": "text/html, charset=UTF-8",
            "longMessage": "long message1"
        }
    ],
    "description": "description3",
    "id": "25"
}
```

21.5.5.5 Notification Template Management: Delete

Request:

- **Operation and URI:** `DELETE http://`*`HOST_NAME`*`:`*`PORT`*`/idaas/im/scim/v1/NotificationTemplates/12`
- **Header:**
  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`

Response:

- **Status:** `HTTP/1.1 204 No Content`

21.5.6 System Property Management

This section provides the following examples of the system property resource:

- View System Properties
- Modify System Properties (PATCH)

21.5.6.1 View System Properties

Request:

- **Operation and URI:** `GET http://`*`HOST_NAME`*`:`*`PORT`*`/idaas/im/scim/v1/SystemProperties?filter=(name eq AllowDisabledManagers)`

Response:

- **Status:** `HTTP/1.1 200 OK`
- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:api:messages:2.0:ListResponse"
    ],
    "totalResults": 1,
    "Resources": [
        {
            "schemas": [

"urn:ietf:params:scim:schemas:oracle:core:2.0:OIG:SystemProperty"
            ],
            "id": "40",
            "meta": {
                "location":
"http://HOST_NAME:PORT/idaas/im/scim/v1/SystemProperties/40",
                "resourceType": "SystemProperty"
            },
            "value": "FALSE",
            "name": "AllowDisabledManagers",
            "displayName": "Is disabled manager allowed"
        }
    ]
}
```

21.5.6.2 Modify System Properties (PATCH)

Request:

- **Operation and URI:** `PATCH http://HOST_NAME:PORT/idaas/im/scim/v1/SystemProperties/27`
- **Header:**
  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`
- **Body:**

```
{
  "schemas":
  [
    "urn:ietf:params:scim:api:messages:2.0:PatchOp"
  ],
  "Operations":
  [
    {
      "op":"replace",
      "path":"displayName",
      "value":"new_displayName_for_systemProperty"
    }
  ]
}
```

Response:

- **Status:** `HTTP/1.1 200 OK`
- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:schemas:oracle:core:2.0:OIG:SystemProperty"
    ],
    "id": "27",
    "meta": {
        "lastModified": "2015-02-05T08:44:04.000-08:00",
        "location":
"http://HOST_NAME:PORT/idaas/im/scim/v1/SystemProperties/27",
        "created": "2015-02-05T08:44:04.000-08:00",
        "resourceType": "SystemProperty"
    },
    "value": "NONE",
    "name": "OIM.ChallengeQuestionModificationURL",
    "displayName": "new_displayName_for_systemProperty"
}
```

21.5.7 Service Provider Configuration Management

Request:

- **Operation and URI:** `GET http://HOST_NAME:PORT/idaas/im/scim/v1/ServiceProviderConfigs`

Response:

- **Status:** `HTTP/1.1 200 OK`
- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:api:messages:2.0:ListResponse"
    ],
    "totalResults": 1,
    "Resources": [
        {
            "schemas": [
                "urn:ietf:params:scim:schemas:core:2.0:ServiceProviderConfig"
            ],
            "patch": {
                "supported": true
            },
            "meta": {
                "location":
"http://HOST_NAME:PORT/idaas/im/scim/v1/ServiceProviderConfigs",
                "resourceType": "ServiceProviderConfig"
            },
            "bulk": {
                "maxPayloadSize": 1048576,
                "supported": false,
                "maxOperations": 1000
            },
            "authenticationSchemes": [
                {
                    "documentationUrl": "http://HOST_NAME/help/httpBasic.htm",
                    "specUrl": "http://www.ietf.org/rfc/rfc2617.txt",
                    "description": "Authentication Scheme using the Http Basic
Standard",
                    "name": "HTTP Basic"
                }
            ],
            "documentationUrl": "http://HOST_NAME",
            "changePassword": {
                "supported": true
            },
            "etag": {
                "supported": false
            },
            "sort": {
                "supported": true
            },
            "filter": {
                "supported": true,
                "maxResults": 200
            }
        }
    ]
}
```

21.5.8 Resource Types Management

Request:

- **Operation and URI:** `GET` `http://`*`HOST_NAME`*`:`*`PORT`*`/idaas/im/scim/v1/ResourceTypes?attributes=name`

Response:

- **Status:** `HTTP/1.1 200 OK`
- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:api:messages:2.0:ListResponse"
    ],
    "totalResults": 15,
    "Resources": [
        {
            "name": "User"
        },
        {
            "name": "PasswordValidator"
        },
        {
            "name": "UserNameRecoverer"
        },
        {
            "name": "Schema"
        },
        {
            "name": "ServiceProviderConfig"
        },
        {
            "name": "Organization"
        },
        {
            "name": "Request"
        },
        {
            "name": "Group"
        },
        {
            "name": "PasswordPolicy"
        },
        {
            "name": "SystemProperty"
        },
        {
            "name": "NotificationTemplate"
        },
        {
            "name": "ResourceType"
        },
        {
            "name": "PasswordResetterWithChallenges"
        },
        {
            "name": "UserNameValidator"
        },
        {
            "name": "UserNameGenerator"
        }
    ]
}
```

21.5.9 Using POST Search

The request is issued with `POST [prefix]/.search`, which is an alternate way of searching for resources. Instead of passing parameters on the URL, the request parameters are passed in the POST body, as shown in the following example.

Request:

- **Operation and URI:** `POST http://`*`HOST_NAME`*`:`*`PORT`*`/idaas/im/scim/v1/`Users/.search
  - **Content-Type:** `application/scim+json`
  - **Authorization:** `Bearer h480djs93hd8`
- **Body:**

```
{
     "schemas":["urn:ietf:params:scim:api:messages:2.0:SearchRequest"],
     "attributes": [ "id", "userName" ],
     "filter": "userType eq \"Employee\" and (emails [type eq \"work\" and
value co \"HOST_NAME\"] or name.givenName co \"doe\")",
     "startIndex":1,
     "count":2,
     "sortBy": "userName",
     "sortOrder": "ascending"
}
```

Response:

- **Body:**

```
{
    "schemas": [
        "urn:ietf:params:scim:api:messages:2.0:ListResponse"
    ],
    "totalResults": 2,
    "itemsPerPage": 2,
    "startIndex": 1,
    "Resources": [
        {
            "id": "10",
            "userName": "John Doe"
        },
        {
            "id": "89",
            "userName": "Mary Doe"
        }
    ]
}
```

21.5.10 Retrieving Schemas

To retrieve the schema supported by Oracle Identity Manager REST service, submit the following SCIM request:

```
GET /Schemas
```

21.6 Securing SCIM Resources

SCIM resources are secured by custom Oracle Web Services Manager (OWSM) policy `oracle/multi_token_noauth_over_ssl_rest_service_policy` created by default during installation or upgrade of Oracle Identity Manager. This policy cannot be changed.

This policy combines the functionality of OWSM predefined policies `oracle/multi_token_rest_service_policy` and `oracle/no_authentication_service_policy`. This policy enforces one of the following authentication polices when a token is sent by the client or allows anonymous when no token is supplied:

- HTTP Basic
- SAML 2.0 Bearer token in HTTP header
- HTTP OAM security
- SPNEGO over HTTP security
- JWT token in HTTP header

  See "Using the JSON Web Token Service" on page 22-1 for information about JWT service for SCIM and REST services.

See the following sections in the *Security and Administrator's Guide for Web Services* for more information about the predefined OWSM policies:

- "oracle/multi_token_rest_service_policy"
- "No Behavior Policies"

22

Using the JSON Web Token Service

After you apply bundle patch 11.1.2.3.161018, Oracle Identity Manager provides a JSON Web Token (JWT) service to simplify the use of Oracle Identity Manager SCIM and REST services.

Note:

- For instructions on downloading and applying the 11.1.2.3.161018 bundle patch, refer to the bundle patch documentation.
- For information about the Token service API that lets your acquire a JWT token using which user can securely access REST end points, see *REST API for Oracle Identity Governance Token Service* in the Oracle Identity Management 11g Release 2 (11.1.2.3.0) documentation library. To do so, navigate to the title *REST API for Oracle Identity Governance Token Service* under the Reference section, at the following URL:

  `http://docs.oracle.com/cd/E52734_01/cross/allbooksdocs.htm`

This section describes how to use the JWT service for Oracle Identity Manager SCIM and REST services. It contains the following sections:

- About the JWT Service
- Authentication Scenarios
- Acquiring and Applying a JWT
- JWT-Based OIM Identity Provider for SCIM-REST Authentication
- End Point and Application Details

22.1 About the JWT Service

To simplify the use of the Oracle Identity Manager SCIM and REST services in various deployment scenarios, a JSON Web Token (JWT) service is available and used with Oracle Identity Manager. The JWT produced by the Oracle Identity Manager token service contains a subject claim for an OIM user signed by the Oracle Identity Manager server, and can be presented for authentication to the OWSM agent protecting the SCIM and REST services API.

The default multi-token OWSM policy protecting the Oracle Identity Manager SCIM and REST services accept various standard authentication mechanisms. This OIM Identity Provider is targeted towards OIM enterprise deployments not already integrated into an authentication domain, or to augmented authentication mechanisms, such as WebSSO, which do not natively support REST-style APIs. This section describes the various authentication scenarios supported in Oracle Identity Manager deployments for access to the SCIM and REST services. In some instances, an application might be deployed as an extension to the Oracle Identity Manager UI, while other cases suppose an application independent of the Oracle Identity Manager UI and consuming the SCIM and REST services to satisfy some identity requirements. The SCIM and REST consumer can be an HTML5/JS application loaded from a web-tier distinct from that in which Oracle Identity Manager is deployed and running in a browser, or the backend of a JSP or .Net application. Another scenario is a mobile application consuming the SCIM and REST services.

22.2 Authentication Scenarios

The following use cases describe various authentication scenarios as a prelude to acquiring a JWT and accessing the SCIM and REST services:

- Stand-alone (no WAM):

  According to this use case, authentication is through OIM Portal. Therefore, there is no Oracle Identity Manager UI authentication. Authentication is either of the following:

  – Directly to the OIM REST API using Oracle Identity Manager as the identity store

  – Some other mechanism, such as OAuth2, in which the subject corresponds to an Oracle Identity Manager username.

- OIM domain is WAM-enabled (OAM, SM, TAM):

  The user has authenticated to the WAM domain, which is protecting Oracle Identity Manager. The application is in possession of an SSO token containing a subject corresponding to an Oracle Identity Manager usernam, or the SSO agent has set a special request header asserting the username.

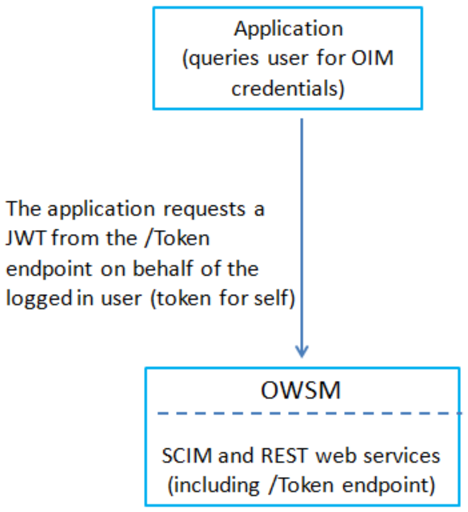
- Mobile application:

  The application will be in possession of a security token, such as OAuth2, containing a subject corresponding to an Oracle Identity Manager username. In all scenarios, OWSM protects the Oracle Identity Manager SCIM and REST services. The OWSM agent is configured with `oracle/multi_token_noauth_over_ssl_rest_service_policy`.

22.3 Acquiring and Applying a JWT

The Oracle Identity Manager SCIM and REST services are augmented with a JWT identity provider presenting a `/iam/governance/token/api/v1/tokens` endpoint, through which the application can acquire a JWT for subsequent use at the SCIM and REST services, as shown in Figure 22–1.

Figure 22–1 Token Endpoint Service


Application
(queries user for OIM credentials)
The application requests a JWT from the /Token endpoint on behalf of the logged in user (token for self)
OWSM
SCIM and REST web services
(including /Token endpoint)

OWSM at the `/iam/governance/token/api/v1/tokens` endpoint authenticates the user and serves as an ID asserter to the `/iam/governance/token/api/v1/tokens` endpoint implementation. The `/iam/governance/token/api/v1/tokens` endpoint can issue a token for self, provided the authenticated user has sufficient privileges. The `/iam/governance/token/api/v1/tokens` implementation validates the target user name, matches an OIM user, and issues a JWT claiming the user name in the HTTP response. This JWT is used for all subsequent access by the application to the Oracle Identity Manager SCIM and REST API.

22.4 JWT-Based OIM Identity Provider for SCIM-REST Authentication

This section describes the JWT-based OIM identity provider for SCIM and REST authentication. It contains the following sections:

- SCIM and REST Security Overview
- JSON Web Token (JWT)
- OIM Identity Provider End Point
- Session Timeout and Refresh

22.4.1 SCIM and REST Security Overview

The SCIM and REST services are protected by an OWSM authentication-only policy. On successful access, the policy establishes a context containing a `javax.security.auth.Subject` containing a `weblogic.security.principal.WLSUserImpl principal`, and the principal contains the authenticated login ID (OIM user name). The SCIM and REST services are deployed in the Oracle Identity Manager server application, and invoke Oracle Identity Manager service APIs, such as `UserManager.createUser` via `OimClient`. The EJB-remoting mechanism converts the WLS login context to an Oracle Identity Manager context, which is subsequently used to identify the caller for authorization and logging purposes.

Note: The following call can be used to retrieve the Subject:

```
Subject activeSubject =
JpsSubject.getSubject(AccessController.getContext());
```

22.4.2 JSON Web Token (JWT)

The JSON Web Token (JWT) is defined in RFC 7519. The JWT issued by OIM Identity Provider contains a claim segment for a subject (OIM user), issuer, expiration, and so on.

```
{"exp":1448420525,"sub":"xelsysadm","iss":"www.oracle.com","prn":"xelsysadm","iat"
:1448418725}
```

The JWT also includes an authentication header (JOSE) indicating the algorithm and certificate key used for the JWS computation:

```
{"alg":"RS256","typ":"JWT","x5t":"8KTfKAncWGbLsNOlLZRQ77qSE74","kid":"xell"}
```

The header and payload are base64url encoded, concatenated, and separated by a ".". Then the RSA signature is computed over that input, base64url encoded, and appended following a ".". The OIM Identity Provider encapsulates the result in another JSON structure, for example:

```
{
    "tokenType": "Bearer",
    "accessToken":
"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsIng1dCI6IjhLVGZLQW5jV0diTHNOT2xMWlJRNzdxU0U3N
CIsImtpZCI6InhlbGwifQ.eyJleHAiOjE0NDg0MjA1MjUsInN1YiI6InhlbHN5c2FkbSIsImlzcyI6Ind3
dy5vcmFjbGUuY29tIiwicHJuIjoieGVsc3lzYWRtIiwiaWF0IjoxNDQ4NDE4NzI1fQ.s6OLNLmYdJXF2Zj
6SaTM5vPHOcKuBIcJlBvVmSATCBKnS-_qmvUYn9-8bcXDbEBo9qum2O3kF0SmbtH0u6-rx-QtNXWupf9-v
btAUVoOpm8f6X3tHVbhzBVixKYnwAZC8tN3LJ6UNOhYzxe7iOZfclmhEQILgA7I3J152gToKmU",
    "expiresIn": "1799"
    }
```

In a subsequent call to the SCIM or REST service, the value of accessToken is submitted in the authentication header, for example JWT, produced via `oracle.security.jps.service.trust.TrustService` and `oracle.security.restsec.jwt.JwtToken`.

22.4.3 OIM Identity Provider End Point

This is a REST service deployed in the OIM server context (JVM) at `/iam/governance/token/api/v1/tokens`.

On successful invocation, it returns a JWT containing a subject claim for an OIM user name. It works in the token-for-self mode, in which the request is processed for the authenticated user; any valid user can request a token for self.

Only POST is supported because allowing GET will expose the user name parameter.

22.4.4 Session Timeout and Refresh

The JWT must include an expiration time, set from a server configuration value. A JWT can be refreshed at the `/iam/governance/token/api/v1/tokens` service by supplying an unexpired JWT using HTTP PUT method. Therefore, OWSM should be configured to accept both HTTP Basic Auth and JWT for the `/iam/governance/token/api/v1/tokens` endpoint.

The HTML5/JS (web browser) application must manage token refresh, including recognizing when the application has been idle and stopping the refresh cycle. After the final token expires, the application will re-authenticate through whatever mechanism was previously used.

A mobile application can securely store credentials necessary to acquire a JWT. Therefore, re-authentication does not necessarily require user interaction, even in the case of username/password authentication.

22.5 End Point and Application Details

This section describes the end points and the application details. It contains the following section:

- End Point

22.5.1 End Point

This section describes the Token end point and Refresh Token end point. It contains the following sections:

- Token End Point (/iam/governance/token/api/v1/tokens)
- Refresh Token Endpoint(/iam/governance/token/api/v1/tokens)

22.5.1.1 Token End Point (/iam/governance/token/api/v1/tokens)

This end point is exposed for application to get the token for user. OWSM at the `/iam/governance/token/api/v1/tokens` endpoint authenticates the user and serves as an ID Asserter to the `/iam/governance/token/api/v1/tokens` endpoint implementation. The `/iam/governance/token/api/v1/tokens` implementation validates that the target user name matches an OIM user and issues a JWT claiming the user name in the HTTP response. This JWT is used for all subsequent access by the application to the Oracle Identity Manager SCIM and REST API.

This end point is enabled on HTTPS only.

Authorization

Consider the following use case for authorization:

Token-For-Self: Any authenticated user can request a token-for-self. This case is identified by an empty payload in the token request. The target user is the user name identified in the Basic Auth header.

Token End Point Scenario

This section describes the following token end point scenario:

Token-For-Self: Web browser via portal with Oracle Identity Manager as the Identity Store

Token Request (Self)

- **Request method:** POST
- **URL:** https://<host>:<ssl port>/iam/governance/token/api/v1/tokens
- **Headers:**
  - **Authorization:** Basic <Base64 encoded user:password>
  - **Accept:** application/json

- **Content-Type:** application/json
- **X-Requested-By:** <random value>

22.5.1.2 Refresh Token Endpoint(/iam/governance/token/api/v1/tokens)

This end point is exposed for application to refresh their existing token by providing old token before they get expired. This end point validates the existing token and then re-issues a token if the existing token is still valid. This end point is protected by JWT OWSM policy.

This end point is enabled on HTTPS only.

Refresh Token End Point Configuration

Table 22–1 shows the Refresh Token end point configuration.

Table 22–1 Refresh Token End Point Configuration

Use Case	Availability	Description
Web browser via portal with Oracle Identity Manager as the Identity Store	Yes	Application gets the token directly by accessing the `/iam/governance/token/api/v1/tokens` end point by providing the user name and password. If the token is about to get expired, then the application should hit the `/iam/governance/token/api/v1/tokens` end point by providing the existing token, and gets a new token.

Refresh Token Request

- **Request method:** PUT
- **URL:** https://<host>:<ssl port>/iam/governance/token/api/v1/tokens
- **Headers:**
  - **Authorization:** Bearer <token value>
  - **Accept:** application/json
  - **Content-Type:** application/json
  - **X-Requested-By:** <random value>

Part VIII

Notification Service

This part contains chapters that describe the notification service and callback service.

It contains the following chapters:

23 Developing Notification Events

This chapter describes how to develop notification events. It contains the following sections:

- Notification Concepts
- Developing Custom Notification

23.1 Notification Concepts

For developing custom notification for various operations in Oracle Identity Manager, the notification engine supports creation of notification events and notification templates.

An event is an operation that occurs in Oracle Identity Manager, such as user creation, request initiation, or any custom event created by the user. The events are generated as part of business operations or via generation of errors. Event definition is the metadata that describes the event. To define metadata for events, it is important to identify all event types supported by a functional component. For example, as a part of the scheduler component, metadata can be defined for scheduled job execution failed and shutting down of the scheduler. Every time a job fails or the scheduler is shut down, the events are raised and notifications associated with that event are sent.

Notification templates are associated to specific events. The templates are used for defining the format of the notification. Oracle Identity Manager provides predefined or default notification templates. In addition, you can create new notification templates.

For some events, Oracle Identity Manager sends notification by default. For example, when a user is created without username and password through UI or reconciliation, the login credentials are notified to the user and user's manager.

You can define new notification events by using notification APIs and resolver class, as described in the subsequent sections. The following are examples of custom notification requirements:

- A user is assigned to a role. The user and role owner are to be notified.
- A user is assigned with a new application instance, which is financially significant, as part of reconciliation. The application instance owner and compliance officer is to be notified for prospective rogue attempts.

23.2 Developing Custom Notification

Developing custom notification is described in the following sections:

- Building the Notification Logic
- Creating Plug-in Pack Containing the Resolver Class
- Building the Invocation Logic
- Configuring the Notification Service

23.2.1 Building the Notification Logic

Building the notification login involves defining the event metadata XML and creating the Resolver class, as described in the following sections:

- Defining Event Metadata
- Creating the Resolver Class
- Creating the plugin.xml File

23.2.1.1 Defining Event Metadata

Corresponding to each event, you must create an XML file that has the specific schema defined by the notification engine. Compliant to that schema (.xsd file), an XML file is created that defines how an event looks like. When the event is defined, you can configure a notification template for that event.

An event file must be compliant with the schema defined by the notification engine, which is NotificationEvent.xsd. The event file contains basic information about the event.

Note: The NotificationEvent.xsd file is in the iam\iam-product\features\notification\metadata directory in the MDS.

The following is a sample event XML file:

```
<?xml version="1.0" encoding="UTF-8"?>
<Events xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../../metadata/NotificationEvent.xsd">
  <EventType name="User Created">
    <StaticData>
      <Attribute DataType="X2-Entity" EntityName="User" Name="Granted User"/>
      <Attribute DataType="X2-Entity" EntityName="User" Name="Grantee User"/>
      <Attribute DataType="91-Entity" EntityName="User Group" Name="User Grp"/>
    </StaticData>
      <Resolver class="oracle.iam.notification.DemoResolver">
      <Param DataType="91-Entity" EntityName="Resource" Name="ResourceInfo"/>
    </Resolver>
  </EventType>
</Events>
```

The event XML file has the following elements:

- **EventType name:** The name of the event that will be available while creating notification templates for the event.

- **StaticData:** The list of static parameters. This set of parameters specifically let the user add parameters that are not data dependent. In other words, this element defines the static data to be displayed when notification template is to be configured. For instance, the user entity is not data dependent, and when resolved, has the same set of attributes for all the event instances and notification templates.
- **Param DataType:** The list of dynamic parameters. This set of parameters specifically let the user add parameters that are data dependent. For instance, the Resource entity is data dependent. Corresponding to this field, a lookup is displayed on the UI. When the user selects the resource object, the call goes to the Resolver class provided to get the fields that are shown in the tree from which user can select the attribute to be used on the template.

  Note: Available data is the list of attributes that can be embedded as a token in the template. These tokens are replaced by the value passed by the resolver class at run time. See step 7 of "Creating a Notification Template" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager* for an example of a token.

  Available data is displayed in a drop-down list while creating a notification template, as described in "Creating a Notification Template" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager*.

  Selected data is a single attribute that helps user to copy and paste the attribute name in a notification template. Selected data is the same attribute name as selected in the Available Data list.

  The dynamic entities supported for lookup are user, resource, and organization. These entity names must be specified in the Param DataType element.

  Note: The <Param DataType> element is not a mandatory element. However, when it is used, the entity names must be specified as User, Resource, or Organization.

- **Resolver class:** The Resolver class must be defined for each notification. It defines what parameters are available in the notification creation screen and how those parameters are replaced when the notification is to be sent. In other words, the resolver class resolves the data dynamically at run time and displays the attributes in the UI. See "Creating the Resolver Class" on page 23-4 for information about implementing the resolver class.

Notification service reads the custom event XML files from the META-INF directory of a plug-in.

The recommended way to use the event XML is by placing it in a plugin's META-INF directory. The structure of the custom notification event plug-in is:

- The lib/ directory
  - Notification_Resolver.jar
- The META-INF directory
  - Notification_Event.xml
- plugin.xml

See "Developing Plug-ins" on page 17-1 for detailed information about creating the plug-in JAR and deploying it by using the Plugin Registration Utility.

23.2.1.2 Creating the Resolver Class

All classes have to implement the NotificationEventResolver interface. This interface provides the following methods:

Note: To compile any custom notification event resolver class, the *ORACLE_HOME*/server/apps/oim.ear/APP-INF/lib/OIMServer.jar file must be included in the CLASSPATH.

The getAvailableData Method

public List<NotificationAttribute> getAvailableData(String eventType, Map<String, Object> params);

This API returns the list of available data variables. These variables are available on the UI while creating or modifying the templates and allows the user to select the variables so that they can be the part of the messages on the template.

The eventType parameter specifies the event name for which the template is to be read.

The params parameter is the map that has the entity name and the corresponding value for which available data is to be fetched. For instance:

map.put("Resource", "laptop");

This helps you fetch the fields associated with the laptop resource or other data according to the code that you have provided in the resolver class.

Sample code:

```
/**
* this is a dummy implementation and uses hardcoded values
* Implementors need to iterate the XML as found through the event type
* params : will have all the specific values that your resolver needs
* for instance resource name = "laptop" that you may want here to be resolved
through your custon implementation
*/

List<NotificationAttribute> list = new ArrayList<NotificationAttribute>();
NotificationAttribute subatr = new NotificationAttribute();
subatr.setName("Dynamic1"); subatr.setType("91-Entity");
subatr.setEntityName("Resource"); subatr.setRequired(false);
subatr.setSearchable(true); subatr.setSubtree(lookup91EntityMetaData("resource"),
params.get(0)); list.add(subatr);
```

The main tree contains the entity information and the subtree contains all the nodes that are available on the UI. The name field from each node in the subtree is available on the UI for selection.

The getReplacedData Method

HashMap<String, String> getReplacedData(String eventType, Map<String, Object> params);

This API returns the resolved value of the variables present on the template at run time when notification is being sent.

The eventType parameter specifies the event name for which the template is to be read.

The params parameter is the map that has the base values, such as usr_key and obj_key, required by the resolver implementation to resolve the rest of the variables in the template.

Sample code:

```
HashMap<String, Object> resolvedData = new HashMap<String, Object>();
resolvedData.put("shortDate", new Date()); resolvedData.put("longDate", new
Date());
String firstName = getUserFirstname(params.get("usr_key"));
resolvedData.put("fname", firstName); resolvedData.put("lname", "lastname");
resolvedData.put("count", "1 million");
return resolvedData;
```

Example: Creating a Custom Resolver Class

Consider the example of Oracle Identity Manager sending email notification to the user who has been added as a proxy. If the requirement is to change the date format in the notification email, then create a new resolver class file, such as AddProxyResolverModified, for notification while adding a proxy. The following is the code for the AddProxyResolverModified resolver class:

```
package oracle.iam.selfservice.notification;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Date;
import java.util.logging.Level;
import java.text.ParseException;
import java.text.SimpleDateFormat;

import static oracle.iam.identity.utils.Constants.PROXY_START_DATE;
import static oracle.iam.identity.utils.Constants.PROXY_END_DATE;
import static oracle.iam.identity.utils.Constants.PROXY_ORIGINAL_USR_NAME;
import static oracle.iam.identity.utils.Constants.PROXY_ORIG_USER_LOGIN;
import static oracle.iam.identity.utils.Constants.FIRSTNAME;
import static oracle.iam.identity.utils.Constants.LASTNAME;
import oracle.iam.notification.impl.NotificationEventResolver;
import oracle.iam.notification.vo.NotificationAttribute;

public class AddProxyResolverModified implements NotificationEventResolver {
public List<NotificationAttribute> getAvailableData(String eventType, Map<String,
Object> params) throws Exception {
return null;
}

    public HashMap<String, Object> getReplacedData(String eventType, Map<String,
Object> params)throws Exception {

        SimpleDateFormat sdfSource = new SimpleDateFormat("MMMMMMMM DD,yyyy
HH:mm:ss a z");
        SimpleDateFormat sdfDestination = new SimpleDateFormat("EEE, d MMM yyyy
HH:mm:ss Z");
        Date sdate = null;
        Date edate = null;

        HashMap<String, Object> resolvedData = new HashMap<String, Object>();
        resolvedData.put("firstName",params.get(FIRSTNAME));
        resolvedData.put("lastName",params.get(LASTNAME));
        resolvedData.put("originalusername",params.get(PROXY_ORIG_USER_LOGIN));


        String proxy_startDate = (String )params.get(PROXY_START_DATE);
        System.out.println("proxy_startDate : " + proxy_startDate);
        String proxy_endDate = (String)  params.get(PROXY_END_DATE);
        System.out.println("proxy_endDate : " + proxy_endDate);

        sdate = sdfSource.parse(proxy_startDate);
        edate = sdfSource.parse(proxy_endDate);


        proxy_startDate = sdfDestination.format(sdate);
        System.out.println("proxy_startDate : " + proxy_startDate);
        proxy_endDate = sdfDestination.format(edate);
        System.out.println("proxy_endDate : " + proxy_endDate);


        resolvedData.put("proxystartdate", proxy_startDate);
        resolvedData.put("proxyenddate", proxy_endDate);
        return resolvedData;
    }
}
```

23.2.1.3 Creating the plugin.xml File

The plugin.xml file is a standard XML file used by the plug-in framework. If any feature uses plug-ins, then it exposes a plug-in point. This XML has the information of plug-in point.

For notification event and resolver plug-in, the exposed plug-in point is:

```
oracle.iam.notification.impl.NotificationEventResolver
```

Sample plugin.xml for Notification event and resolver is:

```
<?xml version="1.0" encoding="UTF-8" ?> <oimplugins
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <plugins pluginpoint="oracle.iam.notification.impl.NotificationEventResolver">
<plugin pluginclass="gov.hhs.notification.resolver.SendChallengeQuestionsResolver"
version="1.0" name="Challenge Question Resolver" />   </plugins>
</oimplugins>
```

23.2.2 Creating Plug-in Pack Containing the Resolver Class

After creating the Resolver class, you must package it into a plug-in JAR file, and deploy the JAR file by using the Plug-in Registration Utility. See "Developing Plug-ins" on page 17-1 for detailed information about creating the plug-in JAR and deploying it by using the Plugin Registration Utility.

23.2.3 Building the Invocation Logic

After building the notification logic by defining event metadata XML and creating the Resolver class, you can call the notification logic at a specific operation in Oracle Identity Manager. This is achieved by using event handlers.

The invocation logic for the notification is built as a custom event handler. The custom event handler is then configured at the right stage in the relevant operation. The custom event handler is then deployed by using the Plugin Registration Utility. See "Developing Event Handlers" on page 18-1 for details about developing event handlers.

23.2.4 Configuring the Notification Service

You have the following options for creating the infrastructure-level configuration for notification:

- **Notification Configuration in Oracle Identity Manager:** In Oracle Identity Manager, notification configurations are handled via notification providers. UMS is the default notification provider. For information about notification providers, see "Managing Notification Providers" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager*.
- **Notification Configuration in BPEL workflow:** SOA exposes notification service, which can be called in BPEL workflow for notification. For information about SOA email notification, see "Configuring SOA Email Notification" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager*.

Note: Dynamic Monitoring Service (DMS) can be used to view performance metrics. The following DMS metrics are present for monitoring notification performance:

- `OIM_Notification`: It provides the fine grained details about the time taken by the notification provider.
- `oracle.iam.notification.api.NotificationService`: It provides details, such as the number of notifications and time taken by notification.

Part IX

Customization Lifecycle

This part describes how to use the customization utilities provided by Oracle Identity Manager.

It contains the following chapter:

24 Deploying and Undeploying Customizations

This chapter contains the following topics:

24.1 Migrating User Modifiable Metadata Files

The user modifiable metadata XML files can be exported to MDS, imported from MDS, and deleted from MDS by using Oracle Enterprise Manager.

This section contains the following topics:

24.1.1 Exporting Metadata Files to MDS

To export metadata XML files to MDS:

1. Login to Oracle Enterprise Manager as the administrator user by navigating to the URL in the following format:

   http://*ADMINSTRATION_SERVER*/em

   Make sure that the Administrative Server and at least one Oracle Identity Manager Managed Server are running.

2. Navigate to **Identity and Access**, **oim**, **oim(*VERSION*)**. Right-click and navigate to **System MBean Browser**.

3. Under Application Defined MBeans, navigate to **oracle.mds.lcm**, **Server:oim_server1**, **Application:OIMMetadata**, **MDSAppRuntime**.

4. Export metadata by using the operations. To do so:

   a. Select and open the first exportMetadata operation in the list.

   b. For toLocation, provide the path to a temporary directory, in which this file is to be exported. This file will be exported to the computer on which Oracle

Identity Manager is running. Therefore, make sure that the directory path you specify exist on that computer.

c. For docs, click the pencil icon, click **Add**, and in the Element box, provide the full path of the file to be exported. By clicking **Add**, you can provide the path to multiple docs. Click **OK** at the bottom after adding the metadata docs to be exported.

d. Invoke the operation.

24.1.2 Importing Metadata Files from MDS

To import metadata XML files from MDS:

1. Login to Oracle Enterprise Manager as the admin user. Make sure that the Administrative Server and at least one Oracle Identity Manager Managed Server are running.
2. Navigate to **Identity and Access**, **oim**, **oim(*VERSION*)**. Right-click and navigate to **System MBean Browser**.
3. Under Application Defined MBeans, navigate to **oracle.mds.lcm**, **Server:oim_server1**, **Application:OIMMetadata**, **MDSAppRuntime**.
4. Import metadata by using the operations. To do so:
   a. In the Operations tab, select the first importMetadata operation in the list.
   b. For fromLocation, provide the directory path of the Oracle Identity Manager host from where documents are to be imported.
   c. For docs, click the pencil icon, click **Add**, and in the Element box, provide the full path of the file to be imported. By clicking **Add**, you can provide the path to multiple docs. If no value is provided, then it imports everything under the fromLocation directory recursively.
   d. Invoke the operation.

24.1.3 Deleting Metadata Files from MDS

To delete metadata XML files from MDS.

1. Navigate to MDSAppRuntime mbeans, as described in step 1 of "Exporting Metadata Files to MDS" on page 24-1.
2. Delete metadata by using the operations. To do so:
   a. In the Operations tab, select the first deleteMetadata operation in the list.
   b. For docs, click the pencil icon, click **Add**, and in the Element box, provide the full path of the file to be deleted. By clicking **Add**, you can provide the path to multiple docs to be deleted.
   c. Invoke the operation.

24.1.4 Creating MDS Backup

You might need to create a backup of the MDS before performing customizations. To create a backup of the MDS by using Oracle Enterprise Manager:

1. Login to Oracle Enterprise Manager as the administrator.

2. Navigate to **Application Deployments**, **oracle.iam.console.identity.self-service.ear(V2.0)**. Right-click and navigate to **MDS configuration**.

3. Under Export, select the **Export metadata documents to an archive on the machine where this web browser is running** option, and then click **Export**.

   All the metadata is exported in a ZIP file.

24.1.5 Exporting All MDS Data for Oracle Identity Manager

Some configurations for Oracle Identity Manager are stored in an MDS repository rather than on a file system on the Oracle Identity Manager Server. Troubleshooting configuration issues can sometimes require exporting all MDS data in order to examine it and make corrections if required.

To export all of the Oracle Identity Manager metadata contained in the MDS repository:

1. Setup the environment as a prerequisite:

   a. To perform MDS operations, log in to the Oracle Identity Manager server host with the account used to install and run WebLogic Application Server.

   b. Set you environment variables for the Oracle Identity Manager domain by running the appropriate `setDomainEnv` script found in the *MIDDLEWAR_HOME*/user_projects/domains/*DOMAIN_NAME*/bin/ directory. The command is as shown:

      ```
      $ cd MIDDLEWARE_HOME/user_projects/domains/OIMDomain/bin
      $ .setDomainEnv.sh
      ```

   c. Create a temporary directory, such as /tmp/OIM/MDSData/, which will be used to store the resulting XML files from the database.

   d. Verify that the application server is up and running.

   e. Ensure that you know the WebLogic administrator username and the URL to the Admin Server.

2. Perform the export, as follows:

   a. In the command shell or console window, go to the *OIM_ORACLE_HOME*/common/bin/ directory.

   b. Run the `wlst.sh` command, and then run the `connect()` command, as shown:

      ```
      $ ./wlst.sh
      CLASSPATH=/opt/oracle/Middleware/wlserver_10.3/server/ext/jdbc/oracle/11g/o
      jdbc6dms.jar:...
      ...
      Your environment has been set.
      ...
      Initializing WebLogic Scripting Tool (WLST) ...

      Welcome to WebLogic Server Administration Scripting Shell

      Type help() for help on available commands
      wls:/offline> connect()
      Please enter your username [weblogic] :
      Please enter your password [welcome1] :
      Please enter your server URL [t3://localhost:port] :
      Connecting to t3://localhost:port with userid weblogic ...
      ```

```
Successfully connected to Admin Server 'AdminServer' that belongs to domain
'OIMDomain'.

Warning: An insecure protocol was used to connect to the server. To ensure
on-the-wire security, the SSL port or Admin port should be used instead.
```

c. Provide the WebLogic administrator username and password and the URL to the Admin Server.

d. Run the `exportMetadata` command providing at least the `application`, `server`, and `toLocation` arguments, as shown:

Note: Be sure to pass the argument data in single quotes, such as:

```
server='oim_server1'
```

```
wls:/OIMDomain/serverConfig> exportMetadata(application='OIMMetadata',
server='oim_server1', toLocation='/tmp/OIM/MDSData')
```

e. A list of the files exported is displayed. At this point, you can run the `disconnect()` command followed by the `exit()` command, as shown:

```
wls:/OIMDomain/serverConfig> disconnect()
Disconnected from weblogic server: AdminServer
wls:/offline> exit()


Exiting WebLogic Scripting Tool.

$
```

f. Go to the /tmp/OIM/MDSData/ directory, and view the db/oim-config.xml file, or the db/form-metadata/FormMetaData.xml file, or any other exported MDS file.

The following is an example WLST script for exporting all MDS files:

```
connect('WEBLOGIC_USERNAME','PASSWORD','t3://localhost:PORT')
exportMetadata(application='OIMMetadata', server='oim_server1',
toLocation='/tmp/OIM/MDSData')
disconnect()
exit()
```

You can save this script in a .py file, for example /tmp/exportOIMMDS.py, which you can run to automatically produce the same results. The following is a sample .py file:

```
cd MIDDLEWARE_HOME/user_projects/domains/OIMDomain/bin
. setDomainEnv.sh
mkdir -p /tmp/OIM/MDSData
cd $OIM_ORACLE_HOME/common/bin
./wlst.sh /tmp/exportOIMMDS.py
```

24.2 Migrating JARs and Resource Bundle

When migrating from test to production environment, all the connector artifacts must be migrated to the respective database tables, which can be done using the following utilities to migrate JAR files and resource bundle:

- Upload JAR Utility
- Download JAR Utility
- Delete JAR Utility
- Upload Resource Bundle Utility
- Download Resource Bundle Utility
- Delete Resource Bundle Utility

Note:

- All the Upload JAR and Resource Bundle utilities must be run from the *OIM_HOME*/bin/ directory.
- Make sure that wlfullclient.jar is generated before running these utilities.
- Set *APP_SERVER*, *OIM_ORACLE_HOME*, *JAVA_HOME*, *MW_HOME*, *WL_HOME*, and *DOMAIN_HOME* before running the scripts.
- All the scripts for the JAR files and resource bundles support both interactive mode and command-line mode usage. But it is recommended to use interactive mode because this is secure and the passwords are not echoed on the console.
- For running the scripts in command-line mode, run it with the -help argument. For example:

  ```
  sh UploadJars.sh -help
  ```

  To upload a JAR file in the silent mode:

  ```
  UploadJars.sh [-username USERNAME] [-password PASSWORD]
  [-serverURL <t3://OIM_HOSTNAME:OIM_PORT>] [-ctxFactory
  <weblogic.jndi.WLInitialContextFactory>] [-JavaTasks
  LOCATION_OF_JAVA_TASK_JAR]
  ```

  For information about configuring the utilities to upload/download JAR files and resource bundle over SSL, see "Configuring SSL for Oracle Identity Manager Utilities" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager*.

  To upload multiple JAR files in the silent mode:

  ```
  UploadJars.sh [-username USERNAME] [-password PASSWORD]
  [-serverURL <t3://OIM_HOSTNAME:OIM_PORT>] [-ctxFactory
  <weblogic.jndi.WLInitialContextFactory>] [-JavaTasks
  LOCATION_OF_JAVA_TASK_JAR] [-ScheduleTask
  LOCATION_OF_SCHEDULED_TASK_JAR] [-ThirdParty
  LOCATION_OF_THIRD_PARTY_JAR] [-ICFBundle
  LOCATION_OF_ICF_BUNDLE_JAR]
  ```

- In this document, interactive mode usage of the JAR and Resource Bundle utilities are explained because it is a secure way of running the utilities and is recommended.

  To run the JAR or Resource Bundle utilities in interactive mode, run the scripts without specifying any arguments. For example:

  ```
  sh UploadJars.sh
  ```

24.2.1 Upload JAR Utility

The UploadJars.sh and UploadJars.bat scripts are available in the *OIM_HOME*/bin/ directory. Running these scripts upload the JAR files in to the database.

A sample invocation of this utility is as shown:

```
[Enter Xellerate admin username :]ADMISTRATOR_LOGIN
[Enter the admin password :]ADMINISTRATOR_PASSWORD
[[Enter serverURL (Ex. t3://oimhostname:oimportno for
weblogic)]:]t3://xyz.com:14000
[[Enter context (Ex. weblogic.jndi.WLInitialContextFactory for
weblogic)]:]weblogic.jndi.WLInitialContextFactory
Enter the jar type
 1.JavaTasks
 2.ScheduleTask
 3.ThirdParty
 4.ICFBundle
1
Enter the path/location of jar file :
/tmp/example.jar
Do u want to load more jars [y/n] :n
```

Note: 14000 is Oracle Identity Manager port.

24.2.2 Download JAR Utility

The DownloadJars.sh and DownloadJars.bat scripts are available in the *OIM_HOME*/bin/ directory. Running these scripts download the JAR files from the database.

A sample invocation of this utility is as shown:

```
[Enter Xellerate admin username :]ADMINISTRATOR_LOGIN
[Enter the admin password :]ADMINISTRATOR_PASSWORD
[[Enter serverURL (Ex. t3://oimhostname:oimport for
weblogic)]:]t3://localhost:14000
[[Enter context (i.e.: weblogic.jndi.WLInitialContextFactory for
weblogic)]:]weblogic.jndi.WLInitialContextFactory
Enter the jar type
1.JavaTasks
2.ScheduleTask
3.ThirdParty
4.ICFBundle
1
Enter the full path of the download directory :
/home/joe/tmp
Enter the name of jar file to be downloaded from DB :
example.jar
Do u want to download more jars [y/n] :n
```

Note: 14000 is Oracle Identity Manager port.

24.2.3 Delete JAR Utility

The DeleteJars.sh and DeleteJars.bat scripts are available at the *OIM_HOME*/bin/ directory. Running these scripts delete the JAR files from the database.

A sample invocation of this utility is as shown:

```
[Enter Xellerate admin username :]ADMINISTRATOR_LOGIN
[Enter the admin password :]ADMINISTRATOR_PASSWORD
[[Enter serverURL (Ex. t3://oimhostname:oimport for
weblogic)]:]t3://localhost:14000
```

```
[[Enter context (i.e.: weblogic.jndi.WLInitialContextFactory for
weblogic)]:]weblogic.jndi.WLInitialContextFactory
Enter the jar type
1.JavaTasks
2.ScheduleTask
3.ThirdParty
4.ICFBundle
1
Enter the name of jar to be deleted from DB :
example.jar
Do u want to delete more jars [y/n] :n
```

24.2.4 Upload Resource Bundle Utility

The UploadResourceBundles.sh and UploadResourceBundles.bat scripts are available in the *OIM_HOME*/server/bin/ directory. Running these scripts upload the connector or custom resources to the database.

A sample invocation of this utility is as shown:

```
Enter Xellerate admin username :]ADMINISTRATOR_LOGIN
[Enter the admin password :]ADMINISTRATOR_PASSWORD
[[Enter serverURL (Ex. t3://oimhostname:oimportno for
weblogic)]:]t3://localhost:14000
[[Enter context (i.e.: weblogic.jndi.WLInitialContextFactory for
weblogic)]:]weblogic.jndi.WLInitialContextFactory
Enter the resource bundle type
 1.Custom Resource
 2.Connector Resource
 2
Enter the path/location of resource bundle file :
/tmp/example.properties
Do u want to load more resource bundles [y/n] :n
```

24.2.5 Download Resource Bundle Utility

The DownloadResourceBundles.sh and DownloadResourceBundles.bat scripts are available in the *OIM_HOME*/bin/ directory. Running these scripts download the resource bundles from the database.

A sample invocation of this utility is as shown:

```
[Enter Xellerate admin username :]ADMINISTRATOR_LOGIN
[Enter the admin password :]ADMINISTRATOR_PASSWORD
[[Enter serverURL (Ex. t3://oimhostname:oimportno for
weblogic)]:]t3://localhost:14000
[[Enter context (i.e.: weblogic.jndi.WLInitialContextFactory for
weblogic)]:]weblogic.jndi.WLInitialContextFactory
Enter the resource bundle type
1.Custom Resource
2.Connector Resource
2
Enter the full path of the download directory :
/home/joe/tmp
Enter the name of resource bundle file :
example.properties
Do u want to download more resource bundles [y/n] :n
```

24.2.6 Delete Resource Bundle Utility

The DeleteResourceBundles.sh and DeleteResourceBundles.bat are available in the OIM_HOME/bin/ directory. Running these utilities delete the resource bundles from the database.

A sample invocation of this utility is as shown:

```
[Enter Xellerate admin username :]ADMINISTRATOR_LOGIN
[Enter the admin password :]ADMINISTRATOR_PASSWORD
[[Enter serverURL (Ex. t3://oimhostname:oimportno for
weblogic)]:]t3://localhost:14000
[[Enter context (i.e.: weblogic.jndi.WLInitialContextFactory for
weblogic)]:]weblogic.jndi.WLInitialContextFactory
Enter the resource bundle type
1.Custom Resource
2.Connector Resource
2
Enter the name of resource bundle file to be deleted from DB:
example.properties
Do u want to delete more resource bundles [y/n] :n
```

Part X

Reports and Audit

This part describes about audit engine and how to configure reports in Oracle Identity Manager.

It contains the following chapters:

25 Understanding BI Publisher in Oracle Identity Manager

Oracle Business Intelligence (BI) Publisher is Oracle's primary reporting tool for authoring, managing, and delivering all highly formatted documents in Oracle Identity Manager. BI Publisher is shipped by default with Oracle Identity Manager 11g Release 2 (11.1.2.3.0).

This chapter describes how to create Oracle Identity Manager reports using embedded BI Publisher, deploy the report, and run it. It contains the following topics:

25.1 Overview

Oracle BI Publisher is an Oracle's enterprise reporting solution and provides a single reporting environment to author, manage, and deliver all of your reports and business documents. Utilizing a set of familiar desktop tools, such as Microsoft Word, Microsoft Excel, or Adobe Acrobat, you can create and maintain report layouts based on data from diverse sources, including Oracle Identity Management products.

See Also: *Oracle Fusion Middleware Developer's Guide for Oracle Business Intelligence Enterprise Edition* to learn more about Oracle BI Publisher functionality.

BI Publisher binaries are installed by default with Oracle Identity Manager in the same Middleware Home directory. Configuration of BI Publisher is accomplished automatically while configuring Oracle Identity Manager.

Oracle BI Publisher provides the following capabilities as part of default installation in Oracle Identity Manager:

- Installs the BI Publisher schema into the Oracle Identity Manager repository database.
- Integrates BI Publisher into the same WebLogic server domain as Oracle Identity Manager. This process configures the primary BI Publisher server named as `bi_server1`.
- Provides highly formatted and professional quality reports with headers/footers.
- Supports PDF, Micorsoft Word, and HTML formats.
- Enables you to develop your own custom reports against the Oracle Identity Manager repository.
- Enables you to use BI Publisher's scheduling capabilities and delivery mechanisms, such as e-mail.

25.2 Benefits of Embedded BI Publisher

As an administrator, configuring embedded BI Publisher have the following benefits:

- In earlier releases of Oracle Identity Manager, BI Publisher is to be downloaded, which is bundled with the Oracle Business Intelligence Enterprise Edition (OBIEE) Suite. Embedded BI Publisher is lightweight and smaller in size than the full BISHIPHOME.
- Embedded BI Publisher can be installed at runtime during Oracle Identity Manager installation, and can be configured within the same WebLogic domain on which Oracle Identity Manager is deployed.
- The BI Publisher reports access is granted to the Oracle Identity Manager user administrator by default via the Oracle Identity Manager role `BIReportAdministrator`. User members of this role can get the access of BI Publisher and Oracle Identity Manager reports.
- By default, Oracle Identity Manager administrative users have access to the BI Publisher and Oracle Identity Manager reports. By logging in to Oracle Identity Manager, administrator can grant the `BIReportAdministrator` role to other Oracle Identity Manager users, as required.

25.3 Verifying the Integration of BI Publisher with Oracle Identity Manager

To verify the integration of BI Publisher with Oracle Identity Manager in fresh configuration mode:

1. Login to BI Publisher by using your Oracle Identity Manager system administrator credentials by navigating to the following URL:

   http://*HOST_NAME*:*PORT*/xmlpserver

   The default port for BI Publisher server is 9704.

   Note: Make sure that BI Publisher server is running when accessing the BI Publisher URL.

2. Click Catalog. The Oracle Identity Manager directory with reports is displayed under the `Shared Folders` directory.

   You can now use the full capabilities of BI Publisher, such as PDF report generation and e-mail delivery.

Note:

- In addition to Oracle Identity Manager System administrator credentials, you can also access BI Publisher by using the WebLogic credentials and BISystemUser credentials.
- By default, BISystemUser password is same as that of Oracle Identity Manager system administrator password.

25.4 Granting BI Publisher Access to Other Oracle Identity Manager Users

The BI Publisher reports access is granted to the Oracle Identity Manager user administrator by default via the Oracle Identity Manager role `BIReportAdministrator`.

All Oracle Identity Manager users that are members of this `BIReportAdministrator` role have access to the reports catalog and administrator tabs of BI Publisher.

By default, only Oracle Identity Manager System administrators has the rights to login to BI Publisher and access Oracle Manager Identity reports.

Note: When BI Publisher is installed, an OPSS application role for it is created. This OPSS application role is combined with the permissions on the BI Publisher catalog to achieve the BI Publisher-specific security rules. For detailed information about OPSS, see *Oracle Fusion Middleware Application Security Guide*.

25.5 Creating and Deploying BI Publisher Reports

For comprehensive information on using BI Publisher to create and deploy reports, see the BI Publisher documentation library at the following URL:

`http://www.oracle.com/technetwork/middleware/bi-publisher/documentation/index.html`

25.6 Configuring SSL-Enabled Email Server

To configure SSL-enabled email server for BI Publisher:

1. Import the target system certificate into the JDK (or JRE) used by Oracle Identity Manager. For example:

   ```
   keytool -import -keystore MY_CACERTS -file CERT_FILE_NAME -storepass PASSWORD
   ```

   In this command:

   - *MY_CACERTS* is the full path and name of the certificate store. The default is `cacerts`.
   - *CERT_FILE_NAME* is the full path and name of the certificate file.
   - *PASSWORD* is the password of the keystore.

   For example:

   ```
   keytool -import -keystore /home/OIM/java/jdk/lib/security/cacerts -file
   /home/target.cert -storepass kspassword
   ```

2. Import the target system certificate into the Oracle WebLogic Server keystore, as shown:

   ```
   keytool -import -keystore WEBLOGIC_HOME/server/lib/DemoTrust.jks -file
   CERT_FILE_NAME -storepass PASSWORD
   ```

   Here, *CERT_FILE_NAME* is the full path and name of the certificate file, and *PASSWORD* is the password of the keystore.

   For example:

   ```
   keytool -import -keystore WEBLOGIC_HOME/server/lib/DemoTrust.jks -file
   /home/target.cert -storepass DemoTrustKeyStorePassPhrase
   ```

3. (Optional) To run SOA in non-SSL mode, remove DemoTrust store references from the SOA environment. To do so:

   a. Modify the MSERVER_HOME to remove the DemoTrust references.

   b. Remove the following references from setDomainEnv.sh:

      ```
      -Djavax.net.ssl.trustStore=$WEBLOGIC_HOME/server/lib/DemoTrust.jks from
      EXTRA_JAVA_PROPERTIES
      ```

   c. Restart the Administration server and the Managed servers.

25.7 Configuring SSO in Access Manager Enabled Environment (Optional)

For an Oracle Identity Manager environment that is integrated with Oracle Access Manager (OAM), which uses the default LDAP sources for authentication, Oracle Identity Manager Administrator does not have the required privilege to access Oracle BI Publisher. A specific BI Publisher role must be granted to the administrator user in the LDAP repository. To do so:

1. Login to Oracle Enterprise Manager by using WebLogic administrator user credentials.
2. Expand **Weblogic Domain**. Right-click ***DOMAIN_NAME***, and select **Security**, **Application Roles**.
3. From the Application Stripe list, select **obi**. Click the search icon adjacent to the Role Name field. The `BISystem` role is displayed.
4. To assign the Oracle Identity Manager administrator user to the BISystem role, select the row and click **Edit**. The Edit Application Role: BISystem page is displayed.
5. Click **Add**. The Add Principal dialog box is displayed. From the Type list, select **User**. Enter the user login as `OracleSystemUser` in the Principal Name field. Click the search icon adjacent to the Display Name field.
6. Select the searched principals name. Click **Ok**. The `OracleSystemUser` user is added as member.
7. Click **OK**.
8. Login to BI Publisher and access Oracle Identity Manager reports.

25.8 Patching Embedded BI Publisher Binaries

Embedded BI Publisher does not have Oracle Universal Installer (OUI). It is installable, and an external OUI can be used to install it. Use the OUI install directory that is created when your product is installed. Use the `-oui_loc` option to point to an OUI directory to apply the Opatch for embedded BI Publisher binaries.

Use the following command as an example to apply Opatch on embedded BI Publisher binaries:

```
opatch apply PATCH_LOCATION
```

26 Understanding Auditing

User profile audits cover changes to user profile attributes, user membership, resource provisioning, access policies, and resource forms.

The audit engine collects auditing information in Oracle Identity Manager. Whenever a profile is modified, the audit engine captures the changes (the delta) and updates (or generates, if missing) the snapshots of the user and role profiles and stores these snapshots and deltas in XML format. The audit engine also contains post-processors, which, based on the generated XML, populate the reporting tables with relevant data. To maintain high performance, by default the audit engine performs these tasks in an asynchronous and offline manner by using the underlying Java Messaging Service (JMS) provided by the application server.

This chapter discusses the following topics:

- Audit Levels
- Tables Used for Storing Information About Auditors
- Issuing Audit Messages

26.1 Audit Levels

As mentioned earlier in this chapter, when you install Oracle Identity Manager user profile auditing is enabled by default and the auditing level is set to Resource Form. If you change the auditing level, then you must run the GenerateSnapshot.sh script (on UNIX) or the GenerateSnapshot.bat script (on Microsoft Windows). This script is in the *IDM_HOME*/server/bin directory. The script examines all users in Oracle Identity Manager database and generates new snapshots based on the new auditing level.

Note: Before running the GenerateSnapshot script, you must set the following environment variables:

- APP_SERVER: Set the value to weblogic.
- OIM_ORACLE_HOME: Set it to the directory on which Oracle Identity Manager is installed.
- JAVA_HOME: Set it to the directory path of the Java Runtime directory for the Oracle Identity Manager server.
- WL_HOME: Set it to the directory on which Oracle WebLogic Server is installed.
- MW_HOME: Set it to the directory on which Oracle Fusion Middleware is installed.
- DOMAIN_HOME: Set it to the Oracle Identity Manager domain.

When you run the GenerateSnapshot script, you are prompted to enter the following:

```
[Enter Xellerate admin username :]SYSTEM_ADMINISTRATOR_USERNAME
[Enter password for xelsysadm :]SYSTEM_ADMINISTRATOR_PASSWORD
[Threads to use [ 8 ]]
[Enter serverURL :[t3://OIM_HOST:OIM_PORT]
[Enter context Factory :[ weblogic.jndi.WLInitialContextFactory]
```

Note: If you change the auditing level, then you must run the GenerateSnapshot script before allowing users to access the system.

You can configure the "level of detail for auditing" aspect of the auditing engine and specify the audit level as the value of the XL.UserProfileAuditDataCollection system property in the Advanced Administration.

See Also: "System Properties in Oracle Identity Manager" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager* for information about this system property

The supported audit levels are:

- **Process Task:** Audits the entire user profile snapshot together with the resource lifecycle process.
- **Resource Form:** Audits user record, role membership, resource provisioned, and any form data associated to the resource.
- **Resource:** Audits the user record, role membership, and resource provisioning.
- **Membership:** Only audits the user record and role membership.
- **Core:** Only audits the user record.
- **None:** No audit is stored.

Note: When you specify a particular audit level, all audit levels that are at a lower priority level are automatically enabled. For example, if you specify the Membership audit level, then the Core audit level is automatically enabled.

Audit level specifications are case-sensitive. When you specify an audit level, ensure that you do not change the case (uppercase and lowercase) of the audit level.

26.2 Tables Used for Storing Information About Auditors

Information about auditors is stored in the following tables of the database:

- **AUD:** This table stores metadata about all the auditors defined in Oracle Identity Manager.
- **aud_jms:** This table stores data to be consumed by the audit engine and eventually by the auditors. It is an operational and intermediate staging table.

  The key in this table is sent to the JMS. Oracle Identity Manager uses this table to control the order of the changes when multiple changes are made to the same user. You can use the Issue Audit Messages Task scheduled task to automate the reissue of messages that are not processed. For more information about this scheduled task, see "Managing the Scheduler" in *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager*.

26.3 Issuing Audit Messages

Oracle Identity Manager provides a scheduled task named Issue Audit Messages Task. This scheduled task retrieves audit message details from the aud_jms table and sends a single JMS message for a particular identifier and auditor entry in the aud_jms table. An MDB processes the corresponding audit message.

The following is the attribute of this task:

Max Records

Use the Max Records attribute to specify the maximum number of audit messages to be processed for a specified scheduled task run. The default value of this attribute is 400.

If there is a backlog of audit messages in the aud_jms table, then you can increase the value of the Max Records attribute. The value that you set depends on how many messages the JMS engine can process during the default scheduled task execution interval. This, in turn, depends on the performance of the application server and database. Before increasing the Max Records value, you must determine how much time is taken to process the number of audit messages in the JMS destination (oimAuditQueue) by, for example, using the administrative console of the application server. If the time taken is less than the scheduled task interval, then you can make a corresponding increase in the value of the Max Records attribute.

Part XI

Appendixes

This part contains the following appendixes:

A The FacesUtils Class

The FacesUtils class is used in the customization use cases shown in "Using Managed Beans" on page 19-38. This class contains various helper methods for re-rendering components, evaluating EL expressions, and accessing attributes through binding.

Example A–1 provides the code snippet of the FacesUtils class with implementation of some of the methods:

Note: If you add the code from Example A–1 in any ViewControllerProject source code, then some packages, such as the following, might not be found:

```
import oracle.adf.model.BindingContext;
import oracle.adf.model.binding.DCBindingContainer;
import oracle.adf.model.binding.DCControlBinding;
import oracle.binding.AttributeBinding;
import oracle.binding.ControlBinding;
```

You must add ADF Model Runtime to the class path to resolve the errors related to importing these packages. To add ADF Model Runtime to project class path:

1. Right-click the project, and select **Project Properties**.
2. On the left navigation bar, select **Libraries and Classpath**.
3. Click **Add Library**.
4. Under Extension, select **ADF Model Runtime**.
5. Click **OK**.
6. Click **OK**.

Example A–1 Sample FacesUtils Class

```
package oracle.iam.ui.sample.common.view.utils;

import java.io.IOException;

import java.util.Map;
import java.util.ResourceBundle;

import javax.el.ELContext;
import javax.el.ExpressionFactory;
import javax.el.MethodExpression;
import javax.el.ValueExpression;
import javax.faces.application.Application;
import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;

import oracle.adf.model.BindingContext;
import oracle.adf.model.binding.DCBindingContainer;
import oracle.adf.model.binding.DCControlBinding;
import oracle.adf.view.rich.context.AdfFacesContext;

import oracle.binding.AttributeBinding;
import oracle.binding.ControlBinding;

import oracle.iam.ui.platform.utils.TaskFlowUtils;

import oracle.jbo.uicli.binding.JUCtrlActionBinding;
import oracle.jbo.uicli.binding.JUCtrlListBinding;
import oracle.jbo.uicli.binding.JUEventBinding;


public class FacesUtils {

    private FacesUtils() {
        // do not instantiate
        throw new AssertionError();
    }

    /*
     * Re-render the component.
     */
    public static void partialRender(UIComponent component) {
        if (component != null) {
            AdfFacesContext.getCurrentInstance().addPartialTarget(component);
        }
    }

    /*
     * Sets attribute value through attribute binding.
     */
    public static void setAttributeBindingValue(String attributeName,
                                                Object value) {
        AttributeBinding binding = getAttributeBinding(attributeName);
        if (binding != null) {
            binding.setInputValue(value);
        } else {
            throw new IllegalArgumentException("Binding " + attributeName +
                                               " does not exist.");
        }
    }

    /*
     * Gets attribute value using attribute binding.
     */
    public static <T> T getAttributeBindingValue(String attributeName,
                                                 Class<T> clazz) {
        AttributeBinding binding = getAttributeBinding(attributeName);
        if (binding != null) {
            return (T)binding.getInputValue();
        } else {
```

```
        throw new IllegalArgumentException("Binding " + attributeName +
                                           " does not exist.");
    }
}

/*
 * Gets attribute value using list binding.
 */
public static <T> T getListBindingValue(String attributeName,
                                        Class<T> clazz) {
    ControlBinding ctrlBinding = getControlBinding(attributeName);
    if (ctrlBinding instanceof JUCtrlListBinding) {
        JUCtrlListBinding listBinding = (JUCtrlListBinding)ctrlBinding;
        return (T)listBinding.getAttributeValue();
    } else {
        throw new IllegalArgumentException("Binding " + attributeName +
                                           " is not list binding.");
    }
}

public static ControlBinding getControlBinding(String name) {
    ControlBinding crtlBinding = getBindings().getControlBinding(name);
    if (crtlBinding == null) {
        throw new IllegalArgumentException("Control Binding '" + name +
                                           "' not found");
    }
    return crtlBinding;
}

public static AttributeBinding getAttributeBinding(String name) {
    ControlBinding ctrlBinding = getControlBinding(name);
    AttributeBinding attributeBinding = null;
    if (ctrlBinding != null) {
        if (ctrlBinding instanceof AttributeBinding) {
            attributeBinding = (AttributeBinding)ctrlBinding;
        }
    }
    return attributeBinding;
}

public static DCBindingContainer getBindings() {
    FacesContext fc = FacesContext.getCurrentInstance();
    ExpressionFactory exprfactory =
        fc.getApplication().getExpressionFactory();
    ELContext elctx = fc.getELContext();

    ValueExpression valueExpression =
        exprfactory.createValueExpression(elctx, "#{bindings}",
                                          Object.class);

    DCBindingContainer dcbinding =
        (DCBindingContainer)valueExpression.getValue(elctx);

    return dcbinding;
}

/*
 * Evaluates EL expression and returns value.
 */
public static <T> T getValueFromELExpression(String expression,
                                               Class<T> clazz) {
        FacesContext facesContext = FacesContext.getCurrentInstance();
        Application app = facesContext.getApplication();
        ExpressionFactory elFactory = app.getExpressionFactory();
        ELContext elContext = facesContext.getELContext();
        ValueExpression valueExp =
            elFactory.createValueExpression(elContext, expression, clazz);
        return (T)valueExp.getValue(elContext);
    }

    /*
     * Gets MethodExpression based on the EL expression. MethodExpression can then
be used to invoke the method.
     */
    public static MethodExpression getMethodExpressionFromEL(String expression,
                                                             Class<?> returnType,
                                                             Class[] paramTypes) {
        FacesContext facesContext = FacesContext.getCurrentInstance();
        Application app = facesContext.getApplication();
        ExpressionFactory elFactory = app.getExpressionFactory();
        ELContext elContext = facesContext.getELContext();
        MethodExpression methodExp =
            elFactory.createMethodExpression(elContext, expression, returnType,
                                             paramTypes);
        return methodExp;
    }

    public static ELContext getELContext() {
        return FacesContext.getCurrentInstance().getELContext();
    }

    /*
     * Shows FacesMessage. The message will not be bound to any component.
     */
    public static void showFacesMessage(FacesMessage fm) {
        FacesContext.getCurrentInstance().addMessage(null, fm);
    }

    /*
     * Launch bounded taskFlow based on provided parameters.
     */
    public static void launchTaskFlow(String id, String taskFlowId,
                                      String name, String icon,
                                      String description, String helpTopicId,
                                      boolean inDialog,
                                      Map<String, Object> params) {
        // create JSON payload for the contextual event
        String jsonPayLoad =
            TaskFlowUtils.createContextualEventPayLoad(id, taskFlowId,
                                                       name, icon, description,
                                                       helpTopicId, inDialog,
                                                       params);

        // create and enqueue contextual event
        DCBindingContainer bc =

(DCBindingContainer)BindingContext.getCurrent().getCurrentBindingsEntry();
        DCControlBinding ctrlBinding =
bc.findCtrlBinding(TaskFlowUtils.RAISE_TASK_FLOW_LAUNCH_EVENT);
        // support both bindings - using eventBinding as well as methodAction
        if (ctrlBinding instanceof JUEventBinding) {
            JUEventBinding eventProducer = (JUEventBinding) ctrlBinding;
            bc.getEventDispatcher().queueEvent(eventProducer, jsonPayLoad);
        } else if (ctrlBinding instanceof JUCtrlActionBinding) {
            JUCtrlActionBinding actionBinding = (JUCtrlActionBinding) ctrlBinding;
            bc.getEventDispatcher().queueEvent(actionBinding.getEventProducer(),
jsonPayLoad);
        } else {
            throw new IllegalArgumentException("Incorrect binding for " +
TaskFlowUtils.RAISE_TASK_FLOW_LAUNCH_EVENT);
        }
        bc.getEventDispatcher().processContextualEvents();
    }

    /*
     * Redirect to a provided url.
     */
    public static void redirect(String url) {
        try {
            FacesContext fctx = FacesContext.getCurrentInstance();
            fctx.getExternalContext().redirect(url);
            fctx.responseComplete();
        } catch (IOException ex) {
            throw new RuntimeException(ex);
        }
    }

}
```

B

Username Reservation and Common Name Generation

This appendix contains the following topics:

- Username Reservation
- Common Name Generation

B.1 Username Reservation

When the request for user creation is submitted, the following scenarios are possible:

- While the request is pending, another create user request is submitted with the same username. If the second request is approved and the user is created, then the first request, when approved, fails because the username already exists in Oracle Identity Manager.
- While the request is pending, another user with the same username is directly created in the LDAP identity store. When the create user request is approved, it fails while provisioning the user entity to LDAP because an entry already exists in LDAP with the same username.

To avoid these problems, you can reserve the username in both Oracle Identity Manager and LDAP while the create user request is pending for approval. If a request is created to create a user with the same username, then an error message is displayed and the create user request is not created.

For reserving the username:

- The USER ATTRIBUTE RESERVATION ENABLED system property must be set to TRUE for the functionality to be enabled. For information about searching and modifying system properties, see "Managing System Properties" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager*.
- Reservation in LDAP is done only if reservation functionality is enabled, and LDAP is in sync with Oracle Identity Manager database. For information about synchronization between Oracle identity Manager and LDAP identity store, see "Integration Between LDAP Identity Store and Oracle Identity Manager" in *Oracle Fusion Middleware Integration Guide for Oracle Identity Management Suite*.

Note:

- If LDAP provider is not configured, then the reservation is done only in Oracle Identity Manager.
- When LDAP synchronization and user attribute reservation features are enabled, it is recommended to enable UID uniqueness in the directory server. Without this, user reservation in the directory does not work properly because while the user is reserved in the reservation container, the user with the same user ID can be created in the user container. This results is user creation failure when Oracle Identity Manager tries to move the user from the reservation container to the user container.

If user attribute reservation is enabled, the reservation happens in two phases:

In the first phase, an entry is created in Oracle Identity Manager database and a user is created in reservation container. This entry in Oracle Identity Manager database is removed after successful creation of user, rejection by approver, or request failure.

In the second phase, in LDAP, on successful creation, the user is moved to the reservation container. In other cases such as rejection by approver or request failure, the user is removed from the reservation container.

After the request-level and operation-level approvals are obtained for the create user request, the username is no longer reserved in the username container in LDAP. The username is moved to the container in which the existing users are stored. The user is also created in Oracle Identity Manager.

This section consists of the following topics:

- Enabling and Disabling Username Reservation
- Configuring the Username Policy
- Writing Custom User Name Policy
- Releasing the Username
- Configuring Username Generation to Support Microsoft Active Directory

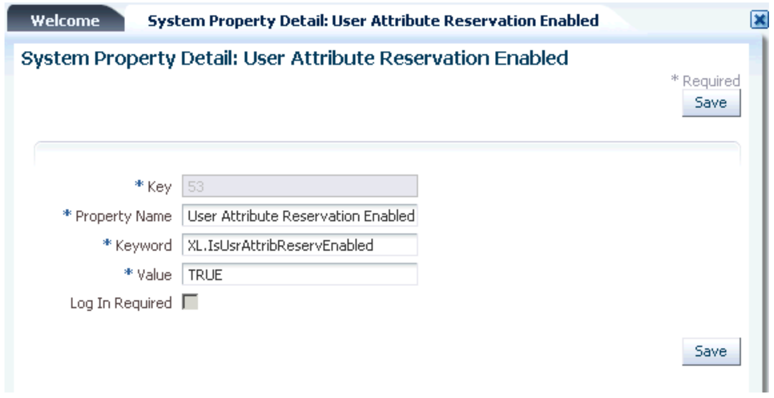
B.1.1 Enabling and Disabling Username Reservation

The username reservation functionality is enabled by default in Oracle Identity Manager. This is done by keeping the value of the USER ATTRIBUTE RESERVATION ENABLED system property to TRUE. You can verify the value of this system property in the System Configuration section of the Oracle Identity Manager System Administration Console.

To disable username reservation:

1. Log in to Oracle Identity System Administration.
2. In the left pane, under System Management, click **System Configuration.** The Advanced Administration opens in a new window.
3. In the left pane, click the search icon to search for all existing system properties. A list of system properties are displayed in the search results table.
4. Click **User Attribute Reservation Enabled**. The System Property Detail page for the selected system property is displayed, as shown in Figure B–1:

Figure B–1 The System Property Detail Page



System Property Detail: User Attribute Reservation Enabled

* Key	53
* Property Name	User Attribute Reservation Enabled
* Keyword	XL.IsUsrAttribReservEnabled
* Value	TRUE
Log In Required	☐

5. In the Value field, enter **False**.
6. Click **Save**. The username reservation functionality is disabled.

B.1.2 Configuring the Username Policy

Username Policy is a plugin implementation for username operations such as username generation and username validation. You can change the default policies from the System Configuration section in Oracle Identity System Administration.

In case of a Create User usecase, the plugins are invoked only if the user login is not provided. In such a case, the plugin to be invoked is picked up from the system property, "Default policy for username generation". The custom user name policy is honored in all the following use cases:

- Create Admin User
- Create User Request
- Reconciliation
- Bulk Load

Table B–1 lists the predefined username policies provided by Oracle Identity Manager. In this table, the dollar ($) sign in the username generation indicates random alphabet:

Table B–1 Predefined Username Policies

Policy Name	Expected Information	Username Generated
oracle.iam.identity.usermgmt.impl.plugins.EmailIdPolicy	E-mail	E-mail value is used as the auto-generated user name
oracle.iam.identity.usermgmt.impl.plugins.LastNameFirstInitialLocalePolicy	First name, last name, and locale	last name + first initial_locale, last name + middle initial + first initial_locale, last name + $ + first initial_locale (all possibilities of single random alphabets), last name + $$ + first initial_locale
oracle.iam.identity.usermgmt.impl.plugins.FirstInitialLastNameLocalePolicy	Firstname, Lastname, Locale	first initial + lastname_locale, first initial + middle initial + first name_locale, first initial + $ + lastname_locale, first initial + $$ + lastname_locale

Table B–1 (Cont.) Predefined Username Policies

Policy Name	Expected Information	Username Generated
oracle.iam.identity.usermgmt.impl.plugins.LastNameFirstInitialPolicy	Firstname, Lastname	lastname+firstInitial, lastname+middleinitial+firstInitial, lastname+$+firstInitial (all possibilities of single random alphabets) , lastname+$$+firstInitial
oracle.iam.identity.usermgmt.impl.plugins.FirstInitialLastNamePolicy	Firstname, Lastname	firstInitial+lastname, firstInitial+middleInitial+firstname, firstInitial+$+lastname, firstInitial+$$+lastname
oracle.iam.identity.usermgmt.impl.plugins.LastNameFirstNamePolicy	Firstname, Lastname	lastname.firstname, lastname.middleinitial.firstname, lastname.$.firstname (all possibilities of single random alphabets) , lastname.$$.firstname
oracle.iam.identity.usermgmt.impl.plugins.FirstNameLastNamePolicy	Firstname, Lastname	firstname.lastname, firstname.middleinitial.lastname, firstname.$.lastname (all possibilities of single random alphabets) , firstname.$$.lastname
oracle.iam.identity.usermgmt.impl.plugins.DefaultComboPolicy	Any one of the following: - Email - Firstname, Last Name - Last name.	If e-mail is provided, then username is generated based on the e-mail. If e-mail is not available, then it generates username based on firstname and lastname by appending a user domain to it. If first name is not available, then it generates the username based of the last name only by appending a user domain to it. The user domain is configured as the Default user name domain system property, and the default value is @oracle.com
oracle.iam.identity.usermgmt.impl.plugins.LastNamePolicy,	Lastname	lastname, middle initial + lastname , $ + lastname, $$ + lastname
oracle.iam.identity.usermgmt.impl.plugins.LastNameLocalePolicy	Lastname, Locale	lastname_locale, middle initial + lastname_locale , $ + lastname_locale, $$ + lastname_locale
oracle.iam.identity.usermgmt.impl.plugins.FirstNameLastNamePolicyForAD	Firstname, Lastname	firstname+lastname, substring of firstname+lastname+$, substring of firstname+ substring of lastname+$
oracle.iam.identity.usermgmt.impl.plugins.LastNameFirstNamePolicyForAD	Lastname, Firstname	lastname+firstname, lastname+substring of firstname+$, substring of lastname+ substring of firstname+$

The policy implementations generate the username, check for its availability, and if the username is not available, then generate other username based on the policy in the order mentioned in Table B–1, and repeat the procedure. The dollar ($) sign in the username generation indicates random alphabet. If any of the expected information is missing, then the policies generate errors.

Values must be provided for all the parameters of the username generation format. If any of the parameters are not provided, then Oracle Identity Manager generates an error. For example, If the firstname.lastname policy is configured and the firstname is not provided, then the error would be "An error occurred while generating the Username. Please provide firstname as expected by the firstname.lastname policy".

The username generation is exposed as public APIs in User Manager. Oracle Identity Manager provides an utility class for accessing the functionality of generating user names. The class that contains utility methods is as shown:

```
oracle.iam.identity.usermgmt.api.UserManager
```

The UserManager class exposes the following public API for username generation and validation:

```
//Method that will generate username based on default policy

    public String generateUserNameFromDefaultPolicy(Map<String,  Object> attrMap)
 throws UserNameGenerationException,  UserManagerException;

//Method that will generate username based on policy

    public String generateUserNameFromPolicy(String policyId,  Map<String, Object>
attrMap) throws UserNameGenerationException,  UserManagerException;

//Method that will check whether username is valid against default policy

    public boolean isUserNameValidForDefaultPolicy(String userName,  Map<String,
Object> attrMap) throws UserManagerException;

//Method that will check whether username is valid against given policy

    public boolean isUserNameValidForPolicy(String userName, String  policyId,
Map<String, Object> attrMap) throws  UserManagerException;

//Method to return all policies (including customer written)

        public List<Map<String, String>> getAllUserNamePolicies(Locale locale)

//Method that will return policy description in given locale

    public String getPolicyDescription(String policyID, Locale locale)
```

Table B–2 lists the constants defined in oracle.iam.identity.usermgmt.utils.UserNameGenerationUtil to represent the policy ID of the default username policies:

Table B–2 Constants Representing Policy IDs

Policy Name	Constant
EmailIDPolicy	EMAIL_ID_POLICY
LastNameFirstInitialLocalePolicy	FIRSTNAME_LASTNAME_POLICY
FirstInitialLastNameLocalePolicy	LASTNAME_FIRSTNAME_POLICY
LastNameFirstInitialPolicy	FIRSTINITIAL_LASTNAME_POLICY
FirstInitialLastNamePolicy	LASTNAME_FIRSTINITIAL_POLICY
LastNameFirstNamePolicy	FIRSTINITIAL_LASTNAME_LOCALE_POLICY
FirstNameLastNamePolicy	LASTNAME_FIRSTINITIAL_LOCALE_POLICY
DefaultComboPolicy	DEFAULT_COMBO_POLICY

Table B–2 (Cont.) Constants Representing Policy IDs

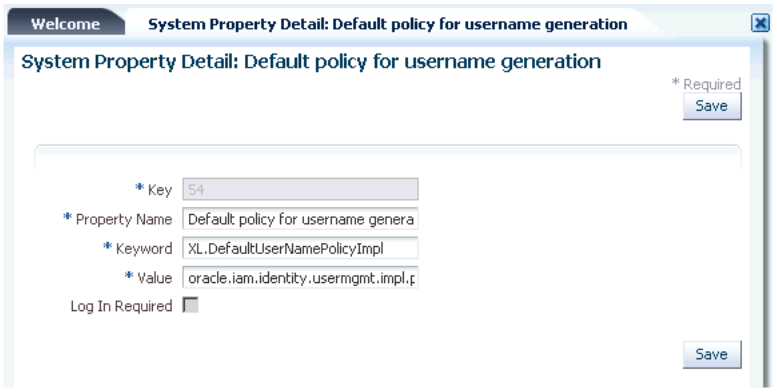
Policy Name	Constant
LastNamePolicy	LASTNAME_POLICY
LastNameLocalePolicy	LASTNAME_LOCALE_POLICY
FirstNameLastNamePolicyForAD	FIRSTNAME_LASTNAME_POLICY_FOR_AD
LastNameFirstNamePolicyForAD	LASTNAME_FIRSTNAME_POLICY_FOR_AD

When called to generate username, the policy classes expect the attribute values to be set in a map by using the key constants defined in the oracle.iam.identity.utils class.Constants. This means that a proper parameter value must be passed to call the method by using the appropriate constant defined for it, for example, the FirstName parameter has a constant defined for it.

The default username policy can be configured by using the Oracle Identity System Administration. To do so:

1. Navigate to the System Configuration section.
2. Search for all the system properties.
3. Click **Default policy for username generation**. The System Property Detail page for the selected property is displayed, as shown in Figure B–2:

Figure B–2 The Default Username Policy Configuration


Welcome
System Property Detail: Default policy for username generation
* Required
Save
Save

System Property Detail: Default policy for username generation

* Key	54
* Property Name	Default policy for username genera
* Keyword	XL.DefaultUserNamePolicyImpl
* Value	oracle.iam.identity.usermgmt.impl.p
Log In Required	☐

The XL.DefaultUserNameImpl system property is provided for picking up the default policy implementation. By default, it points to the default username policy, which is oracle.iam.identity.usermgmt.impl.plugins.DefaultComboPolicy displayed in the Value field.

4. In the Value field, enter **oracle.iam.identity.usermgmt.impl.plugins.*POLICY***. Here, *POLICY* is one of the policy implementations.

Note: All the plug-ins must be registered with Oracle Identity Manager by using the /identity/metadata/plugin.xml file. A sample plugin.xml file is as shown:

```
<plugins
pluginpoint="oracle.iam.identity.usermgmt.api.UserNamePolicy">
<plugin
pluginclass="oracle.iam.identity.usermgmt.impl.plugins.LastNameFirs
tNamePolicy"
version="1.0" name="LastNameFirstNamePolicy"/>
</plugins>
```

5. Click **Save**.

B.1.3 Writing Custom User Name Policy

You can write your own policies by adding new plug-ins and changing the default policies from the System Configuration section in Oracle Identity System Administration.

See Also: "Developing Plug-ins" on page 17-1 for information about the plug-in framework

The UserManager exposes APIs for username operations. The APIs take the user data as input and return a generated username. The APIs make a call to plug-ins that return the username. This allows you to replace the default policies with custom plug-ins with your implementation for username operations.

Note:

- For user name generation and validation, public APIs are exposed in UserManager.
- While creating the policy, ensure that the attributes used in generating the username are defined in the request data set.

You can write your own username policies by implementing the plug-in interface, as shown:

```
package oracle.iam.identity.usermgmt.api;

public interface UserNameGenerationPolicy extends
 oracle.iam.identity.usermgmt.api.UserNamePolicy {
public String getUserName(Map<String, Object> reqData) throws
UserNameGenerationException;
public boolean isGivenUserNameValid(String userName, Map<String, Object> reqData);

//methods inherited from old user name policy interface
//oracle.iam.identity.usermgmt.api.UserNamePolicy
public String getUserNameFromPolicy(HashMap<String, String> reqData) throws
UserNameGenerationException;
public boolean isUserNameValid(String userName, HashMap<String, String> reqData);
public String getDescription(Locale locale);

}
```

This plug-in point is exposed as a kernel plug-in that takes request data as input and returns the username. Each plug-in expects some information and generates username based on that information provided.

Note: Oracle Identity Manager provides an abstract implementation of the oracle.iam.identity.usermgmt.api.UserNameGenerationPolicy interface as the oracle.iam.identity.usermgmt.api.AbstractUserNameGenerationPolicy class name. Therefore, you need not implement the following two methods:

```
public String getUserNameFromPolicy(HashMap<String, String>
reqData) throws UserNameGenerationException;

public boolean isUserNameValid(String userName, HashMap<String,
String> reqData);
```

Create the plug-in ZIP with lib (containing the JAR) and plugin.xml, and then register it by performing the procedure in section "Registering and Unregistering Plug-ins By Using the Plugin Registration Utility" on page 17-8. The following is a sample plugin.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<oimplugins>
<plugins pluginpoint="oracle.iam.identity.usermgmt.api.UserNamePolicy">
<plugin
pluginclass="oracle.iam.identity.usermgmt.impl.plugins.CustomDepartmentNumberEmplo
yeeNumberPolicy" version="1.0" name="CustomDepartmentNumberEmployeeNumberPolicy"/>
</plugins>
</oimplugins>
```

The following are the guidelines on while writing custom user name policies:

- Policies should implement the new interface oracle.iam.identity.usermgmt.api.UserNameGenerationPolicy.
- Custom user name policies must be re-entrant. This means that the custom code in the policy should return the same user login if approver has updated an attribute that does not contribute in generating the user login.

For sample implementation please refer below:

```
package oracle.iam.identity.usermgmt.impl.plugins;

import java.util.Locale;
import java.util.Map;

import oracle.iam.identity.exception.UserNameGenerationException;
import oracle.iam.identity.usermgmt.api.AbstractUserNameGenerationPolicy;
import oracle.iam.identity.usermgmt.api.UserManagerConstants;
import oracle.iam.identity.usermgmt.api.UserNameGenerationPolicy;

public class CustomDepartmentNumberEmployeeNumberPolicy extends
AbstractUserNameGenerationPolicy implements UserNameGenerationPolicy {


          private String departmentNumberKey =
UserManagerConstants.AttributeName.DEPARTMENT_NUMBER.getId();

          private String employeeNumberKey =
UserManagerConstants.AttributeName.EMPLOYEE_NUMBER.getId();

          @Override
          public String getUserName(Map<String, Object> reqData)
                                throws UserNameGenerationException {

          String departmentnumber = reqData.get(departmentNumberKey) == null ?
null : reqData.get(departmentNumberKey).toString();

          String employeeNumber = reqData.get(employeeNumberKey) == null ? null :
reqData.get(employeeNumberKey).toString();

          // Required in case of approver edit. If approver has not modified any
attribute which contributes in user name generation , then return same old user
login

          //Check if user data is not changed using checkForSameUserLogin method
present in AbstractUserNameGenerationPolicy, then return same user login

          //OR use Map<String, Object> existingData = (Map<String, Object>)
reqData.get(oracle.iam.identity.usermgmt.api.UserManagerConstants.EXISTING_DATA )
to implement your own comparison logic

          // If existingData is NULL, it means generate a new user login. If it is
not NULL, then it means policy is invoked during approver edit.

          // If it is NOT NULL, Compare value of participating attributes from
existingData and reqData. If same, return same user login as present in
existingData ; otherwise generate a new user login.

          String oldUserLogin = checkForSameUserLogin(reqData , new
String[]{departmentNumberKey , employeeNumberKey});
               if(oldUserLogin!=null)
                    return oldUserLogin;

               // TODO: DO basic validations. Also, Ensure newly generated user
name is unique and not reserved. You may use utility methods in
oracle.iam.identity.usermgmt.utils.UserNamePolicyUtil for preforming validations.
               return departmentnumber + "-" + employeeNumber;
          }

          @Override
          public boolean isGivenUserNameValid(String userName, Map<String, Object>
reqData) {
               // TODO : custom implementation
               return true;
}

@Override
public String getDescription(Locale locale) {
               return "User Name Generation Policy using department number and
employee number";
     }

}
```

B.1.4 Releasing the Username

The username is released in the following scenarios:

- When the request is approved, and the user is successfully created in Oracle Identity Manager and provisioned to LDAP, and the username from the reserved table is removed. The reserved username is removed after successful user creation after the approvals. The reserved entry in LDAP is removed and the actual user is created.
- If the request is rejected, then the reserved entry of username in LDAP and Oracle Identity Manager is removed.
- If the request fails while or before creating a user in Oracle Identity Manager or LDAP, then the reserved username is deleted.

B.1.5 Configuring Username Generation to Support Microsoft Active Directory

In Oracle Identity Manager deployment with LDAP synchronization is enabled, where Microsoft Active Directory (AD) is the data store, the User Login attribute in Oracle Identity Manager is mapped to the uid attribute in LDAP, which in turn is mapped to the sAMAccountName attribute. The sAMAccountName attribute is used as login for all AD-based applications. There is a limitation on the maximum length supported for value contained in the sAMAccountName attribute in AD. It cannot exceed 20 characters.

Oracle Identity Manager accepts user name as an input at the time of user creation and it can be more than 20 characters. Because AD does not support user name of more than 20 characters, Oracle Identity Manager can be configured to generate the user name, which consists of less than 20 characters.

When AD is used as data store, you can configure the autogeneration of user name by setting the value of the XL.DefaultUserNamePolicyImpl system property to any one of the following:

- **FirstNameLastNamePolicyForAD:** Generates the user login by prefixing a substring from the first name to that of the last name
- **LastNameFirstNamePolicyForAD:** Generates the user login by prefixing a substring from last name to that of the first name

See "Administering System Properties" for information about the XL.DefaultUserNamePolicyImpl system property and setting values of system properties.

Note: If AD is the data store, then any one of the FirstNameLastNamePolicyForAD or LastNameFirstNamePolicyForAD policies must be used. Any other user name generation policy will fail to generate the user name.

B.2 Common Name Generation

The generation of the Common Name user attribute value in Oracle Identity Manager is described in the following sections:

- Common Name Generation for Create User Operation
- Common Name Generation for Modify User Operation

B.2.1 Common Name Generation for Create User Operation

In an LDAP-enabled deployment of Oracle Identity Manager, Fusion applications such as Human Capability Management (HCM) does not pass the common name via SPML request. Given that the common name is a mandatory attribute in LDAP and Oracle Identity Manager is setup to use it as the RDN, Oracle Identity Manager must generate a unique common name.

Based on the description on Common Name, it is the user's display name consisting of first name and last name. Therefore, Oracle Identity Manager generates the Common Name with the help of a common name generation policy that specifies the Common Name in the "firstname lastname" format.

To configure common name generation in Oracle Identity Manager, set the value of the XL.DefaultCommonNamePolicyImpl system property to oracle.iam.identity.usermgmt.impl.plugins.FirstNameLastNamePolicy. For information about the XL.DefaultCommonNamePolicyImpl system property and setting the value of a system property, see "Managing System Properties" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager*.

The following are the details of the FirstNameLastNamePolicy:

- Expected information: Firstname, Lastname
- Common Name generated: firstname.lastname, firstname.$.lastname (all possibilities of single random alphabets), firstname.$$.lastname and so on until a unique common name is generated

Note: The common name must be reserved until the user is created by the request so that multiple requests generated simultaneously having same first and last names do not generate the same common name.

B.2.2 Common Name Generation for Modify User Operation

When the user profile is modified, one or more attributes can change. HCM cannot filter out and send only the modified data to Oracle Identity Manager because it does not have the old user attributes and cannot determine which ones are modified. Therefore, all attributes including the common name (CN) are passed to Oracle Identity Manager by the SPML request. Because the CN changed, Oracle Identity Manager attempts a modify operation (modrdn) in the directory resulting in DN change. Because of this unintended DN change, the group membership DN becomes stale resulting in the user loosing membership in that group. This subsequently results in authorization failure. This happens when referential integrity is turned off in the LDAP server, and therefore, the referenced groups are not updated when the RDN of the user changes. Therefore, referential integrity must be turned on in the target LDAP server. Otherwise, the group memberships become stale. The referential integrity issue is also applicable to roles. Groups are also members of other groups and any RDN changes must be reflected as well.

You can turn on the referential integrity by setting the value of the XL.IsReferentialIntegrityEnabled system property to TRUE. For information about this system property, see "Managing System Properties" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Manager*.

Table B–3 lists the possible scenarios when RDN is modified:

Table B–3 RDN Modification Scenarios

Referential Integrity in LDAP	XL.IsReferentialIntegrity Enabled	Result of Modify Operation (modrdn)
Disabled	FALSE	Oracle Identity Manager generates an error and operation fails.
Disabled	TRUE	Modify operation passes from Oracle Identity Manager and RDN is changed in LDAP. However, the group references are not updated and are stale. This configuration is not recommended.
Enabled	FALSE	Oracle Identity Manager generates an error and modify operation fails. This property must be set to TRUE in Oracle Identity Manager because referential integrity is enabled in LDAP.
Enabled	TRUE	Modify operation passes and RDN is updated. In addition, the references for the DN are updated in LDAP.
Multiple directories with roles and users stored in separate directories. Referential integrity property is not relevant here.	FALSE	Modify operation fails from Oracle Identity Manager. This is not supported by LDAP. Therefore, FALSE is the recommended value in Oracle Identity Manager for the property.
Multiple directories with roles and users stored in separate directories. Referential integrity property is not relevant here.	TRUE	Modify operation passes and RDN is modified. However, because LDAP does not support referential integrity in multiple directories, the group references are stale and must be manually updated.

C Configuring Reports

This appendix describes how to configure Oracle Identity Manager reports when standalone BI Publisher is deployed. Skip this appendix if you are using the embedded reporting capabilities of Oracle Identity Manager, as described in "Understanding BI Publisher in Oracle Identity Manager" on page 25-1.

This appendix contains the following topics:

C.1 What are Oracle Identity Manager Reports?

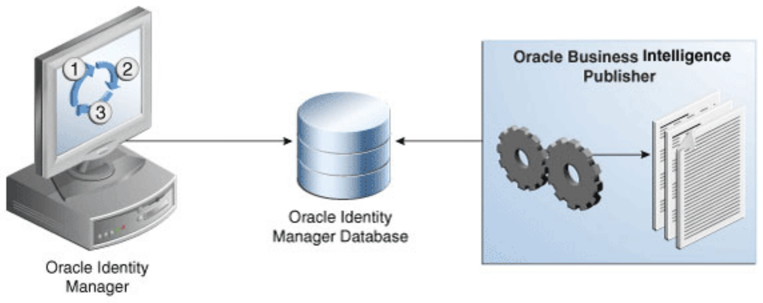
Oracle Identity Manager reports enable you to use Oracle BI Publisher as the reporting solution for Oracle Identity Management products.

Note: Oracle Identity Manager reports are classified based on the functional areas, for instance, Access Policy Reports, Request and Approval Reports, Password Reports, and so on. It is no longer named Operational and Historical.

Oracle Identity Manager reports provide a restricted-use license for Oracle BI Publisher and easy-to-use reporting packages for multiple Oracle Identity Management products.

As shown in Figure C–1, Oracle Identity Manager reports use Oracle BI Publisher to query and report on information in Oracle Identity Management product databases. With minimal setup, Oracle Identity Manager reports provide a common method to create, manage, and deliver Oracle Identity Manager reports.

Figure C–1 Oracle Identity Manager Reports Architecture


1
2
3
Oracle Identity Manager
Oracle Identity Manager Database
Oracle Business Intelligence Publisher

The report templates included in Oracle Identity Manager reports are standard Oracle BI Publisher templates. However, you can customize each template to change its look and feel. If schema definitions for an Oracle Identity Management product are available, you can use that information to create your own custom reports.

Note: Oracle strongly recommends creating back-up copies of the original default report templates before customizing them.

C.2 What is Oracle BI Publisher?

Oracle BI Publisher is an Oracle's enterprise reporting solution and provides a single reporting environment to author, manage, and deliver all of your reports and business documents. Utilizing a set of familiar desktop tools, such as Microsoft Word, Microsoft Excel, or Adobe Acrobat, you can create and maintain report layouts based on data from diverse sources, including Oracle Identity Management products.

See Also: *Oracle Business Intelligence Publisher Documentation* to learn more about Oracle BI Publisher functionality.

C.3 Licensing

Oracle Identity Manager can be separately licensed, independent of any Oracle Application Server or WebLogic edition. BI Publisher is included when you separately license Oracle Identity Manager:

- Shipped BI Publisher reports. Layout changes are allowed, AND
- Shipped or newly created BI Publisher reports that are modified to access data from the existing Identity Management schema that has not been customized.

C.4 Deploying Oracle Identity Manager Reports

This section explains how to deploy Oracle Identity Manager reports 11*g* Release 2 (11.1.2.3.0) and contains the following topics:

- Creating the Metadata Repository
- Installing BI Publisher 11g (11.1.1.7.1)

Note: To use reports on Oracle Identity Manager 11*g* Release 2 (11.1.2.3.0), you must install Oracle BI Publisher 11*g* (11.1.1.7.1). To install Oracle BI Publisher 11*g* (11.1.1.7.1), first install Oracle BI Publisher 11*g* (11.1.1.7.0), and then apply the patch for Oracle BI Publisher 11*g* (11.1.1.7.1) by using OPATCH. The patch number for Oracle BI Publisher 11*g* (11.1.1.7.1) is 16556157, which can be downloaded at the following URL:

`https://support.oracle.com`

You can download Oracle BI Publisher 11g (11.1.1.7.0) by navigating to the following URL:

`http://www.oracle.com/technetwork/middleware/bi-enterprise-edition/downloads/bi-downloads-1923016.html`

See *Oracle Fusion Middleware Installation Guide for Oracle Business Intelligence* for information about installing BI Publisher 11g (11.1.1.7.0).

C.4.1 Creating the Metadata Repository

Each Oracle Business Intelligence system (BI domain) requires its own set of database schemas. Two or more systems cannot share the same set of schemas or repositories.

You must create a repository in your database by using the Repository Configuration Utility (RCU) before installing BI Publisher 11*g* (11.1.1.7.1). For this, you need the RCU utility, which you can download from the Oracle Web site by using the following URL:

`http://www.oracle.com/technetwork/middleware/bi-enterprise-edition/downloads/bi-downloads-1923016.html`

For installing BI Publisher 11*g* (11.1.1.7.1), the following metadata repositories are required:

- Matadata Store (MDS)
- Business Intelligence Platform (BI Platform)

To create the repository in your database by using the RCU utility:

1. Log in to the database as SYSDBA.

   To run RCU, you must have the DBA privilege. Therefore, you must log in as SYSDBA, for example, as user SYS.

2. Navigate to the *RCU_HOME*/bin/ directory.

3. To start RCU:

   - For UNIX, run:

     ```
     ./rcu
     ```

   - For Microsoft Windows, run:

     ```
     rcu.bat
     ```

4. In the Welcome screen that is displayed, click **Next**. The Repository Creation Utility wizard is displayed.

5. In step 1 of the wizard, select **Create**, and then click **Next**. Step 2 of 7: Database Connection Details page is displayed.

6. Specify the connection details, as listed in the following table:

Field	Data to Enter
Database Type	Oracle Database
Host Name	Name of the host on which the database is deployed.
Port	Port number to connect to the host identified in the Host Name field.
Service Name	A string that is the global database name, a name comprised of the database name and domain name, entered during installation or database creation. If you are not sure what the global database name is, then you can obtain it from the combined values of the SERVICE_NAMES parameter in the database initialization file, which is INITSID.ORA. For example, a service name can be SALES.COM, where SALES is the database name and COM is the domain.
Username	Username for a database schema user that has access to Oracle Identity Manager, such as SYS.
Password	Password for the user identified in the Username field.
Role	The role with DBA privilege, such as SYSDBA.

7. Click **Next**. Step 3 of 7: Component Detail page is displayed.
8. Select the Oracle Business Intelligence component. This action automatically selects the MDS schema under the AS Common Schemas group, which is also required by Oracle Business Intelligence.
9. Click **Next**. Step 4 of 7: Schema Passwords page is displayed.
10. Specify the same password for schemas.
11. Click **Next**. Step 5 of 7: Map TableSpaces page is displayed.
12. Click **Next**. A message is displayed after the validation is complete.
13. Click **OK**. Step 6 of 7: Summary page is displayed with the details about the component, schema owner, tablespace type, and tablespace name.
14. Click **Next**. Step 7 of 7: Completion Summary page is displayed.
15. Click **Close**. The metadata repository is created in your database.

Tip: The log files are saved in the *RCU_HOME*\log\ directory.

C.4.2 Installing BI Publisher 11*g* (11.1.1.7.1)

All Oracle Business Intelligence products run on Oracle WebLogic Server domains. Therefore, Oracle WebLogic Server must be installed and configured before you install BI Publisher 11*g* (11.1.1.7.1).

If you do not have Oracle WebLogic Server installed, then OBIEE 11*g* installs the WebLogic Server by default, and creates the bi domain named bifoundation_domain under the user_projects/domains directory. See "Deploying Oracle Identity Manager Reports on OBIEE Environment" on page C-7 for information about deploying Oracle Identity Manager reports on OBIEE environment.

Installing BI Publisher 11*g* (11.1.1.7.1) is described in the following sections:

- Starting the Oracle Business Intelligence Wizard
- Installing BI Publisher 11g (11.1.1.7.1) When Oracle WebLogic Server is Not Installed
- Installing BI Publisher 11g (11.1.1.7.1) When Oracle WebLogic Server is Installed

- Deploying Oracle Identity Manager Reports on OBIEE Environment

Starting the Oracle Business Intelligence Wizard

To start the Oracle Business Intelligence wizard, go to the bishiphome/Disk1/ directory, and run the following:

For UNIX:

```
./runInstaller
```

For Microsoft Windows:

setup.exe

Installing BI Publisher 11g (11.1.1.7.1) When Oracle WebLogic Server is Not Installed

After starting the Oracle Business Intelligence wizard, perform the following steps:

1. In Step 1 of 15: Welcome page of the wizard that is displayed, click **Next**. Step 2 of 15: Type Install page is displayed.
2. Select the **Install Software Updates** option, and click **Next**. Step 3 of 15: Select Installatation Type page is displayed.
3. Select the **Enterprise Install** option, and click **Next**. Step 4 of 15: Pre-requisite Check page is displayed.
4. Click **Next**. Step 5 of 15: Create or Scale BI System page is displayed.
5. Select the **Create New BI System** option. Then, enter values in the following fields:

   User Name: Enter the WebLogic user name.

   Password: Enter a password for the WebLogic user.

   Confirm Password: Re-enter the password for the WebLogic user.

   Doman Name: Name of the WebLogic domain, which is bifoundation_domain by default.
6. Click **Next**. Step 6 of 15: Specify Install Location page is displayed.
7. Enter the directory paths for the installation, and click **Next**.
8. Click **Next**. Step 7 of 12: Configure Components page is displayed.

   Note: When you are installing BI Publisher 11*g* (11.1.1.7.1), the wizard in Configure Components page displays the following options:

   - Business Intelligence Enterprise Edition
   - Business Intelligence Publisher
   - Real Time Decisions
   - Essbase Suite

   If you are using the report function for Oracle Identity Manager 11*g*, then ensure that you select the **Business Intelligence Publisher** option only, and not any other option.
9. Click **Next**. Step 8 of 12: Database Details page is displayed.

10. Enter the details of your database with the credentials that you have specified in the Step 5 of "Creating the Metadata Repository" on page C-3.
11. Complete the remaining steps of the wizard by clicking **Next**.

Installing BI Publisher 11g (11.1.1.7.1) When Oracle WebLogic Server is Installed

When BI Publisher 11g (11.1.1.7.1) is already installed, perform the following steps in the Oracle Business Intelligence wizard:

1. In Step 2 of 15: Type Install page, select the **Simple Install** option, and click **Next**. Step 3 of 15: Prerequisite Check page is displayed.
2. Click **Next**. Step 4 of 12: Middleware Home page is displayed.
3. Enter the Middleware home directory path without trailing space, for example, /u01/app/ Oracle_IDM1/Middelware/.
4. Click **Next**. Step 5 of 12: Administrator Details page is displayed.
5. Enter the administrator user name and password. This account is used for the administration of the WebLogic Server and Enterprise Manager.
6. Click **Next**. Step 6 of 12: Configure Components page is displayed.

Note: When you are installing BI Publisher 11*g* (11.1.1.7.1), the wizard in Configure Components page displays the following options:

- Business Intelligence Enterprise Edition
- Business Intelligence Publisher
- Real Time Decisions
- Essbase Suite

If you are using the report function for Oracle Identity Manager 11*g*, then ensure that you select the **Business Intelligence Publisher** option only, and not any other option.

7. Click **Next**. Step 7 of 12: Database Details page is displayed.
8. Enter the details of your database with the credentials that you have specified in the Step 5 of "Creating the Metadata Repository" on page C-3.
9. Complete the remaining steps of the wizard by clicking **Next**.

Note: If you intend to use the Software Only Install type as part of your strategy to later extend an existing Oracle WebLogic Server domain, then ensure the following:

- The domain is empty, that is, no other software products exist in the domain.
- The domain physically exists on the same computer where you plan to install and configure Oracle Business Intelligence. Extending domains remotely is not supported.

For more information, see "Installing Oracle Business Intelligence" at the following URL:

http://docs.oracle.com/cd/E23943_01/bi.1111/e10539/c4_installing.htm#BIEIG366

Deploying Oracle Identity Manager Reports on OBIEE Environment

To deploy Oracle Identity Manager reports on OBIEE environment:

1. Copy the oim_product_BIP11gReports_11_1_2_3_0.zip file to the BI Publisher directory.
2. Extract the zip file and check for the Oracle Identity Manager and Translations directories created.
3. To sync the catalog with /analytics:
    a. Login to BI Publisher. so that Oracle Identity Manager reports are displayed in the catalog folder on BIP.
    b. Click **Administration**.
    c. In System Maintenance, click `Server Configuration`.
    d. In Catalog, select the `Oracle BIEE Catalog` catalog type, and click **Apply**.
    e. Restart the BI Server.

C.5 Configuring Oracle Identity Manager Reports

This section describes configuring BI Publisher 11*g* (11.1.1.7.1) in the following topics:

- Configuring Security on BI Publisher 11g (11.1.1.7.1)
- Configuring Data Sources for Running Oracle Identity Manager Reports

C.5.1 Configuring Security on BI Publisher 11*g* (11.1.1.7.1)

To configure security in BI Publisher 11*g* (11.1.1.7.1):

1. Login to BI Publisher. To do so:
    a. In a Web browser, enter the URL in the following format:

       http://*HOST_NAME*:*PORT_NUMBER*/xmlpserver/

       For example, http://localhost:7001/xmlpserver/
    b. In the Oracle BI Publisher login page, enter the username and password with WebLogic privileges.
2. Select **Administration, Security Centre, Security Configuration,** and **Local Superuser.**
3. Enable Local Superuser and register an authorized superuser. For example, `xelsysadm`.
4. Restart the BI Publisher Server (bi_server1).
5. Ensure that you are able to log in to the BI Publisher as a registered superuser. For example, xelsysadm.
6. Copy the oim_product_BIP11gReports_11_1_2_3_0.zip file to the BI Publisher directory.
7. Extract the zip file and check for the Oracle Identity Manager and Translations directories created.
8. Navigate to Home and Catalog Folders in the BI Publisher. You can see the Oracle Identity Manager folder under the Shared Folders option.
9. Select **Administration, Data Sources,** and **JDBC Configuration.**

10. In the JDBC Configuration location, create two data sources for OIM 11*g* using the following details:

    - Data Source Name: `OIM JDBC`
      - Driver Type : `Oracle 11g`
      - Database Driver Class : `oracle.jdbc.OracleDriver`
      - Connection String : `jdbc:oracle:thin:@oimhost:1521:orcl`
      - Username : `DEV_OIM`
      - Password: <*PASSWORD*>
    - Data Source Name: `BPEL JDBC`
      - Driver Type : `Oracle 11g`
      - Database Driver Class : `oracle.jdbc.OracleDriver`
      - Connection String : `jdbc:oracle:thin:@oimhost:1521:orcl`
      - Username : `DEV_SOAINFRA`
      - Password: <*PASSWORD*>

C.5.2 Configuring Data Sources for Running Oracle Identity Manager Reports

For Oracle Identity Manager reports, JDBC connections described in the following sections are required:

- Configuring Oracle Identity Manager JDBC Connection
- Configuring BPEL-Based JDBC Connection

C.5.2.1 Configuring Oracle Identity Manager JDBC Connection

To configure Oracle Identity Manager JDBC connection:

1. Click the **Administration** link on the top of the Home page. The BI Publisher Administration page is displayed.
2. Under Data Sources, click the **JDBC Connection** link. The Data Sources page is displayed.
3. In the JDBC tab, click **Add Data Source** to create a JDBC connection to your database. The Add Data Source page is displayed.
4. Enter values in the following fields:
   - **Data Source Name:** Specify the Oracle Identity Manager JDBC connection name, for example, OIM JDBC.
   - **Driver Type:** Select a driver type to suit your database. For example, you can select Oracle 10*g* or Oracle 11*g* to suit your database.
   - **Database Driver Class:** Specify a driver class to suit your database, such as oracle.jdbc.driver.OracleDriver.
   - **Connection String:** Specify the database connection details in the format jdbc:oracle:thin:@*HOST_NAME*:*PORT_NUMBER*:*SID*. For example, jdbc:oracle:thin:@localhost:7003:orcl.
   - **User name:** Specify the Oracle Identity Manager database user name.
   - **Password:** Specify the Oracle Identity Manager database user password.

5. Click **Test** to verify the connection, and then click **Apply** to establish the connection.
6. If the connection to the database is established, a confirmation message is displayed indicating the success. Click **Apply**.

   In the JDBC page, you can see the newly defined Oracle Identity Manager JDBC connection in the list of JDBC data sources.

C.5.2.2 Configuring BPEL-Based JDBC Connection

In BI Publisher, only one data source can be assigned to a report. The first data source is the Oracle Identity Manager data source. The following reports have a secondary data source, which connects to the BPEL database to retrieve BPEL data:

- Task Assignment History
- Request Details
- Request Summary
- Approval Activity

To configure a secondary data source for BPEL-based reports:

1. In the BI Publisher Home page, click **Administration**. The BI Publisher Administration page is displayed.
2. Under Data Sources, click the **JDBC Connection** link. The Data Sources page is displayed.
3. In the JDBC tab, click **Add Data Source** to create a JDBC connection to your database. The Add Data Source page is displayed.
4. Enter values in the following fields:
   - **Data Source Name:** Specify the BPEL JDBC connection name, for example, BPEL JDBC.
   - **Driver Type:** Select a driver type to suit your database. For example, you can select Oracle 10*g* or Oracle 11*g* to suit your database.
   - **Database Driver Class:** Specify a driver class to suit your database, such as oracle.jdbc.driver.OracleDriver.
   - **Connection String:** Specify the database connection details in the format jdbc:oracle:thin:@*HOST_NAME*:*PORT_NUMBER*:*SID*. For example, jdbc:oracle:thin:@localhost:7003:orcl.
   - **User name:** Specify the SOA database user name.
   - **Password:** Specify the SOA database user password.
5. Click **Test** to verify the connection, and then click **Apply** to establish the connection.
6. If the connection to the database is established, a confirmation message is displayed indicating the success. Click **Apply**.

   In the JDBC page, you can see the newly defined BPEL JDBC connection in the list of JDBC data sources.

C.6 Generating Oracle Identity Manager Reports

This section explains how to generate Oracle Identity Manager reports and contains the following topics:

- Generating Sample Reports Against the Sample Data Source
- Generating Reports Against the Oracle Identity Manager JDBC Data Source
- Generating Reports Against the BPEL-Based JDBC Data Source

Note: BI Publisher cannot be accessed through the Oracle Identity Self Service or Oracle Identity System Administration. You must open BI publisher explicitly to access the Oracle Identity Manager 11*g* reports.

C.6.1 Generating Sample Reports Against the Sample Data Source

If you want to see an example of what report data will look like without running a report against the production JDBC Data Source, you can generate a sample report against the Sample Data Source. You must create the Sample Data Source before you can generate sample reports. Refer to appropriate section for your Oracle Identity Management product in "Configuring Oracle Identity Manager Reports" on page C-7 for information on creating the Sample Data Source.

After you create the Sample Data Source you can generate sample reports against it by performing the following steps:

1. Login to Oracle BI Publisher. See step 3 in Configuring Security on BI Publisher 11g (11.1.1.7.1) for more information about logging in to Oracle BI Publisher.
2. Click **Shared Folders**, **Oracle Identity Manager Reports**, and then select **Sample Reports**.
3. Click **View** for the sample report you want to generate.
4. Select an output format for the sample report and click **View**.

   The sample report is generated.

C.6.2 Generating Reports Against the Oracle Identity Manager JDBC Data Source

To generate reports against the Oracle Identity Manager JDBC data source:

1. Log in to Oracle BI Publisher. See step 3 in Configuring Security on BI Publisher 11g (11.1.1.7.1) for more information about logging in to Oracle BI Publisher.
2. Navigate to Oracle Identity Manager reports. To do so:
   a. In the BI Publisher Home page, under Browse/Manage, click **Catalog Folders**. Alternatively, you can click **Catalog** at the top of the page.

      The Catalog page is displayed with a tree structure on the left side of the page and the details on the right.

   b. On the left pane, expand **Shared Folders**, and navigate to Oracle Identity Manager. All the objects in the Oracle Identity Manager folder are displayed.

      You are ready to navigate to BI Publisher 11g and use the Oracle Identity Manager BI Publisher reports.
3. Click **View** for the report you want to generate.

4. Select an output format for the report and click **View**.

   The report is generated.

See Also: *Oracle Business Intelligence Publisher Documentation* to learn more about Oracle BI Publisher.

C.6.3 Generating Reports Against the BPEL-Based JDBC Data Source

The following four reports have a secondary data source, which connects to the BPEL database to retrieve BPEL data:

- Task Assignment History
- Request Details
- Request Summary
- Approval Activity

These reports have a secondary data source, which is the BPEL-based JDBC Data Source, and is called `BPEL JDBC`.

To generate reports against the BPEL-based JDBC data source:

1. Ensure that a BPEL data source exists in BI Publisher. This BPEL Data Source must point to the BPEL database. See "Configuring BPEL-Based JDBC Connection" on page C-9 for more information about creating a BPEL data source.
2. Log in to Oracle BI Publisher. See step 3 in Configuring Security on BI Publisher 11g (11.1.1.7.1) for more information about logging in to Oracle BI Publisher.
3. Navigate to Oracle Identity Manager reports. To do so:
   a. In the BI Publisher Home page, under Browse/Manage, click **Catalog Folders**. Alternatively, you can click **Catalog** at the top of the page.

      The Catalog page is displayed with a tree structure on the left side of the page and the details on the right.

   b. On the left pane, expand **Shared Folders**, and navigate to Oracle Identity Manager. All the objects in the Oracle Identity Manager folder are displayed.

      You are ready to navigate to BI Publisher 11g and use the Oracle Identity Manager BI Publisher reports.
4. Click **Open** for the report you want to generate.
5. Select an output format for the report, and click **Apply**.

   The report is generated based on the BPEL-based JDBC data source.

C.7 Configuring Certification Reports

Certification reports are implemented in Oracle BI Publisher. When using a standalone deployment of BI Publisher, Oracle Identity Manager reports must be deployed on BI Publisher.

Note:

- Oracle Identity Manager reports must be deployed on BI Publisher. See "Generating Certification Reports" in the *Performing Self Service Tasks with Oracle Identity Manager* for information about the default certification reports and generating certification reports.
- If BI Publisher credentials and URL are not configured in Oracle Identity Manager, then the Reports tab in the Dashboard and the Export to PDF or Excel option in the Certification page are not available.

To configure BI Publisher URL:

1. Login to Oracle Enterprise Manager.
2. Click **Identity and Access**.
3. Select **OIM cluster**, **OIM node**, **System MBean Browser**.
4. In the System MBean Browser, navigate to **Application Defined MBeans**, **oracle.iam**, **Server: oim server**, **Application: oim**, **XMLConfig**, **Config**, **XMLConfig.DiscoveryConfig**, **Discovery**.
5. Update the value of the BIPublisherURL attribute with BI Publisher URL.
6. Click **Apply**.

To configure BIP credentials in Oracle Identity Manager:

1. Log in to Oracle Enterprise Manager.
2. In left pane, expand **Weblogic Domain**. The domain name is displayed.
3. Right-click the domain name, and navigate to **Security**, **Credentials**. A list of maps in the credential store, including the oim map, is displayed.
4. Expand the oim map. A list of entries of type Password is displayed.
5. Create a new CSF entry in the oim credential store map, as follows:
   - Select Map: oim
   - Key: BIPWSKey
   - Type: Password
   - Username: *ADMINISTRATOR_USER_NAME*
   - Password: *ADMINISTRATOR_PASSWORD*
   - Description: Login credentials for BI Publisher web service.

After configuring BIP credentials and URL, administration needs to go certification configuration screen (in system administration) and check the "Enable Certification Reports".

To configure the display of the Reports tab in the Detailed Information section of the Dashboard:

1. Log in to Oracle Identity Self Service.
2. Click the **Compliance** tab.

3. Click the **Identity Certification** box, and select **Certification Configuration**. The Certification Configuration page is displayed.
4. Select the **Enable Certification Reports** option.
5. Click **Save**.

Reports can be generated in the following formats:

- PDF
- RTF
- HTML
- Microsoft Excel
- CSV