Oracle® Identity Manager

Audit Report Developer's Guide

Release 9.0.3

B32456-01

February 2007

ORACLE®

Oracle Identity Manager Audit Report Developer's Guide, Release 9.0.3

B32456-01

Primary Author: Don Gosselin

Contents

Preface

The *Oracle Identity Manager Audit Report Developer's Guide* introduces you to the process of generating historical and operational reports related to the audit features of Oracle Identity Manager.

> **Note:** This is a transitional release following Oracle's acquisition of Thor Technologies. Some parts of the product and documentation still refer to the original Thor company name and Xellerate product name and will be rebranded in future releases.

Audience

This document is for Oracle Identity Manager administrators and users. It is assumed that you are familiar with the Oracle Identity Manager system and documentation (specifically, the *Oracle Identity Manager Administrative and User Console Guide*).

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documents

This guide assumes that you have read and understood the following documents:

For more information, see the following documents in the Oracle Identity Manager documentation set:

- *Oracle Identity Manager Installation Guide for JBoss*
- *Oracle Identity Manager Installation Guide for WebLogic*
- *Oracle Identity Manager Installation Guide for WebSphere*
- *Oracle Identity Manager Best Practices Guide*
- *Oracle Identity Manager Globalization Guide*
- *Oracle Identity Manager Design Console Guide*
- *Oracle Identity Manager Administrative and User Console Guide*
- *Oracle Identity Manager Administrative and User Console Customization Guide*
- *Oracle Identity Manager Tools Reference Guide*
- *Oracle Identity Manager API Usage Guide*
- *Oracle Identity Manager Glossary of Terms*

Documentation Updates

Oracle is committed to delivering the best and most recent information available. For information about updates to the Oracle Identity Manager 9.0 documentation set, visit Oracle Technology Network at

`http://www.oracle.com/technology/documentation/index.html`

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.

1

Introduction to Oracle Identity Manager Auditing

Oracle Identity Manager provides audit and compliance reporting. You can use the audits and reports to capture and archive entity and transaction data for compliance monitoring and for IT-centric process and forensic auditing. Oracle Identity Manager, with the audit and compliance modules, provides user profile audits, reports, and attestation.

Oracle Identity Manager auditing consists of historical data, a reporting engine, and a an interface. Archived data identifies users, the information that the users can access, the purpose for the access privileges, and the means for providing the information. You can capture, transport, store, retrieve, and remove historical data over its lifecycle. Security is maintained at every part of the data lifecycle.

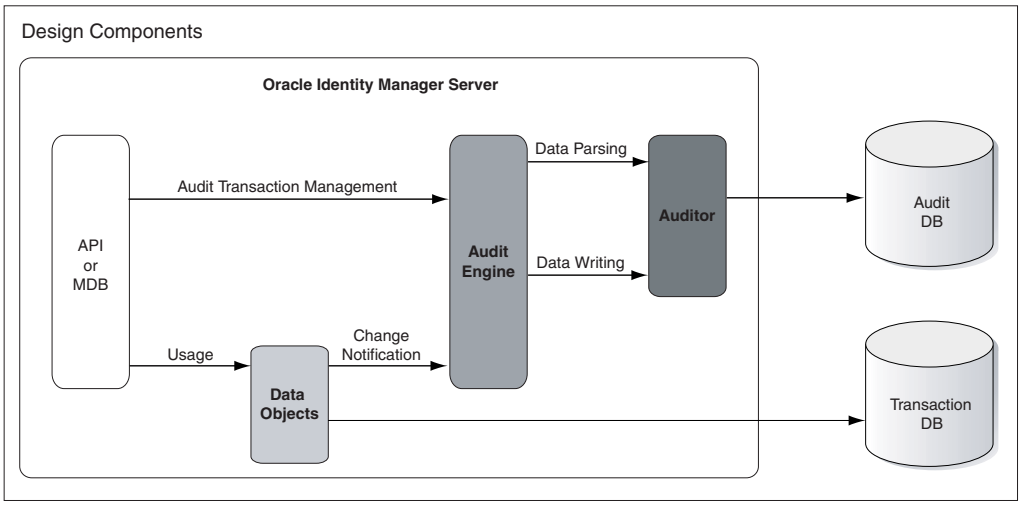
This guide discusses user profile auditing and reporting. See the *Oracle Identity Manager Administrative and User Console Guide* for attestation details.

This chapter discusses the following topics:

Auditing Design Components

Figure 1–1 shows the design components of the Oracle Identity Manager auditing process.

Figure 1–1 Design Components of the Auditing Process



Any action that a user takes in Oracle Identity Manager translates into an application programming interface (API) call or into an MDB picking up a message to process an action.

One action can cause multiple changes. All changes are combined into an *audit transaction*. Each API method that can modify data objects calls the `startTransaction` method in the audit engine at the beginning of the API and the `endTransaction` method at the end of the method call. This defines boundaries for the audit transaction. The audit engine generates a transaction ID to identify the changes made in the transaction.

User Profile Auditing

Oracle Identity Manager provides auditing and historical archiving of a user profile. It takes a snapshot of a user profile, stores the snapshot in an audit table in the database, and updates the snapshot each time the user data changes.

Standard and Customized Reports

Oracle Identity Manager includes standard reports for displaying archived data. You can also create customized reports.

Secondary Data Source Reporting

When you first install Oracle Identity Manager, it uses a primary data source for creating reports. To reduce the load on the primary data source, you can configure a secondary data source for reporting. To use a secondary database, you need to configure replication of data between the transactional data and the reporting database.

2

Oracle Identity Manager User Profile Auditing

User profile audits contain information about changes to user profile attributes, user membership, resource provisioning, access policies, and resource forms.

This chapter discusses the following topics:

Data Collected for User Profile Audits

By default, user profile auditing is enabled and the auditing level is set to `Resource Form` when you install Oracle Identity Manager with the Audit and Compliance module. The auditing level specifies the minimum level required for attestation on form data.

You configure the audit level in the System Properties page of the Administrative and User Console, using the keyword `XL.UserProfileAuditDataCollection`. See "Audit Levels" on page 2-6 for details.

This section discusses the following topics:

Capture and Archiving of User Profile Data

Oracle Identity Manager takes a snapshot of a user profile, stores the snapshot in an audit table in the database, and updates the snapshot each time the user data changes. A snapshot contains all previous data for the user. In Release 9.0.3, Oracle Identity Manager generates a snapshot when an audit is created for a user, even if an initial snapshot is missing. The current snapshot is treated as the initial snapshot.

The following are the contents of a user profile and the tables that make up these components.

- User Record: `USR` table, including all User Defined Files (UDFs)

- User Group Membership: USG, UGP, and RUL information
- User Policy Profile: UPP and UPD information

> **Note:** After changing a group name using the Administrative and User Console, the User Profile Audit tables in the database are not updated with the change until the next snapshot of the user.

User Resource Profile, which consists of:

- User Resource Instance: OIU, OBI , OST, and OBJ information
- Resource Lifecycle (Provisioning) Process: ORC, PKG, TOS, STA, OSI, SCH , MIL
- Resource State (Process) Form: UD_* (including child tables)

XML Representation of the User Profile Snapshot

Oracle Identity Manager stores snapshot data as XML. XML provides a readable format for information about the events that caused a snapshot update. The following sections describe the XML in the snapshot and changes to the field in a User Profile Audit (UPA) table.

Snapshot XML File

The snapshot XML file describes all attributes related to a user profile. The topmost element in this file is UserProfileSnapshot. This element contains a user key and a version for each XML entry. Each subsequent element stores information about the user profile, as follows:

- UserInfo: General information about the user profile
- GroupMembership: Information about group membership
- PolicyProfile: Information about the policy that allowed provisioning for a specific resource
- ResourceProfile: Information about all provisioned resources
  - ResourceInstance: Information about each resource that is provisioned to the user
  - ProcessData: Information about the data stored in the UDFs

The following code snippet is an example of an XML snapshot.

Example 2–1 Snapshot Code Snippet

```
<?xml version="1.0" encoding="UTF-8"?>
- <UserProfileSnapshot key="202" version="1.0">
- <UserInfo>
    <Attribute name="Users.First Name">Testing02First</Attribute>
    <Attribute name="Users.Role">Full-Time</Attribute>
    <Attribute name="Users.Disable User">0</Attribute>
    <Attribute name="Users.Email">amol@thortech.com</Attribute>
    <Attribute name="Users.Status">Active</Attribute>
    <Attribute name="Users.Update Date">2007-01-05 17:12:25.181</Attribute>
    <Attribute name="Users.User ID">TESTING02USER9</Attribute>
    <Attribute name="Users.Xellerate Type">End-User</Attribute>
    <Attribute name="Users.Last Name">Testing02Last</Attribute>
    <Attribute name="Users.Provisioned Date">2007-01-05 17:11:56.868</Attribute>
    <Attribute encrypted="true" name="Users.Password"
```

```
                password="true">8YxO3YSKDXJLmcsKeZhUSw == </Attribute>
    <Attribute name="Users.Creation Date">2007-01-05 17:11:56.868</Attribute>
    <Attribute name="Users.Lock User">0</Attribute>
    <Attribute key="1" name="Users.Updated By Login">XELSYSADM</Attribute>
    <Attribute name="Users.Password Reset Attempts Counter">0</Attribute>
    <Attribute key="1" name="Organizations.Organization Name">Xellerate Users
    </Attribute>
    <Attribute name="Users.Login Attempts Counter">0</Attribute>
    <Attribute key="1" name="Users.Created By Login">XELSYSADM</Attribute>
  </UserInfo>
- <GroupMembership>
-   <Group key="3">
      <Attribute name="Groups-Users.Creation Date">2007-01-05 17:12:30.299
      </Attribute>
      <Attribute name="Groups-Users.Update Date">2007-01-05 17:12:30.299
      </Attribute>
      <Attribute name="Groups-Users.Membership Status">Active</Attribute>
      <Attribute key="1" name="Groups-Users.Updated By Login">XELSYSADM
      </Attribute>
      <Attribute name="Groups-Users.Membership Type">Direct</Attribute>
      <Attribute key="3" name="Groups.Group Name">ALL USERS</Attribute>
      <Attribute key="1" name="Groups-Users.Created By Login">XELSYSADM
      </Attribute>
    </Group>
  </GroupMembership>
- <PolicyProfile>
-   <Policy key="1">
      <Attribute name="UPD_ALLOW_LIST">Res2</Attribute>
      <Attribute name="Access Policies.Key">1</Attribute>
      <Attribute name="Access Policies.Name">AP2</Attribute>
    </Policy>
  </PolicyProfile>
- <ResourceProfile>
-   <ResourceInstance key="57">
      <Attribute name="Users-Object Instance For User.Creation Date">2007-01-05
                17:12:36.599 </Attribute>
      <Attribute key="45" name="Objects.Object Status.Status">Enabled</Attribute>
      <Attribute key="1" name="Access Policies.Name">AP2</Attribute>
      <Attribute key="6" name="Objects.Name">Res2</Attribute>
      <Attribute name="Users-Object Instance For User.Provisioned By Method">
                Access Policy</Attribute>
      <Attribute key="1"
                name="Users-Object Instance For User.Provisioned By Login">
           XELSYSADM</Attribute>
      <Attribute name="Users-Object Instance For User.Provisioned By ID">1
      </Attribute>
      <Attribute key="AP2" name="Access Policies.Key">1</Attribute>
-     <ProcessData>
-       <Parent key="8">
-         <FormInfo>
            <Attribute key="8" name="Structure Utility.Table Name">UD_RES2_PP
             </Attribute>
             <Attribute key="0" name="Structure Utility.Structure Utility Version
                        Label.Version Label">Initial Version</Attribute>
          </FormInfo>
-         <Data key="54">
            <Attribute name="UD_RES2_PP_B">bbbbbbbbbbbb</Attribute>
            <Attribute name="UD_RES2_PP_A">aaaaaaaaaaaa</Attribute>
            <Attribute key="1" name="Access Policies.Name">AP2</Attribute>
          </Data>
```

```
        </Parent>
-       <Children>
-         <Child key="9">
-           <FormInfo>
             <Attribute key="9" name="Structure Utility.Table Name">UD_RES2_CP
              </Attribute>
             <Attribute key="0" name="Structure Utility.Structure Utility Version
                                Label.Version Label">Initial Version</Attribute>
           </FormInfo>
-          <Data key="63">
             <Attribute name="UD_RES2_CP_C">Entry1C</Attribute>
             <Attribute name="UD_RES2_CP_D">Entry1D</Attribute>
             <Attribute key="1" name="Access Policies.Name">AP2</Attribute>
           </Data>
         </Child>
       </Children>
     </ProcessData>
   </ResourceInstance>
-  <ResourceInstance key="74">
     <Attribute name="Users-Object Instance For User.Creation Date">2007-01-05
                  17:22:37.597</Attribute>
     <Attribute key="33" name="Objects.Object Status.Status">Provisioning
     </Attribute>
     <Attribute key="5" name="Objects.Name">Res1</Attribute>
     <Attribute name="Users-Object Instance For User.Provisioned By Method">
                 Direct Provision</Attribute>
     <Attribute key="1" name="Users-Object Instance For User.Provisioned By
                 Login"> XELSYSADM</Attribute>
     <Attribute name="Users-Object Instance For User.Provisioned By ID">
                 XELSYSADM</Attribute>
     </ResourceInstance>
   </ResourceProfile>
 </UserProfileSnapshot>
```

XML File for Changes to a Snapshot

Changes to the original snapshot are stored in an XML file. The information in this file describes all changes that affect user profile attributes for a given transaction. The topmost element in this file is `Changes`. All changes that were made during a particular transaction appear under each `Change` element. The `where` attribute identifies the location of the change. The audit engine makes the changes to an earlier snapshot XML file.

The following is an XML file with snapshot changes:

Example 2–2 Snapshot Changes File

```
<?xml version="1.0" encoding="UTF-8"?>
- <Changes>
-   <Change action="insert" order="1"
    where="/UserProfileSnapshot/ResourceProfile/ResourceInstance[@key='74']">
-     <Attribute name="Users-Object Instance For User.Creation Date">
        <OldValue />
        <NewValue>2007-01-05 17:22:37.597</NewValue>
      </Attribute>
-     <Attribute name="Objects.Object Status.Status">
        <OldValue key="" />
        <NewValue key="35">Ready</NewValue>
      </Attribute>
-     <Attribute name="Objects.Name">
```

```
        <OldValue key="" />
        <NewValue key="5">Res1</NewValue>
      </Attribute>
-     <Attribute name="Users-Object Instance For User.Provisioned By Method">
        <OldValue />
        <NewValue>Direct Provision</NewValue>
      </Attribute>
-     <Attribute name="Users-Object Instance For User.Provisioned By Login">
        <OldValue key="" />
        <NewValue key="1">XELSYSADM</NewValue>
      </Attribute>
-     <Attribute name="Users-Object Instance For User.Provisioned By ID">
        <OldValue />
        <NewValue>XELSYSADM</NewValue>
      </Attribute>
     </Change>
-   <Change action="update" order="2"
     where="/UserProfileSnapshot/ResourceProfile/ResourceInstance[@key='74']">
-     <Attribute name="Objects.Object Status.Status">
        <OldValue key="35">Ready</OldValue>
        <NewValue key="33">Provisioning</NewValue>
      </Attribute>
    </Change>
  </Changes>
```

Information from the UPA table is normalized into `UPA_USR`, `UPA_FIELDS`, `UPA_RESOURCE`, and `UPA_GRP_MEMBERSHIP` for ease of querying for reporting purposes.

Storage of the User Profile Snapshot

When Oracle Identity Manager takes a snapshot of a user profile, it stores the snapshot in a UPA table. The structure of this table is as described in Table 2–1.

Table 2–1 Audit Table Structure

Field	Value
Entry ID	Key for the audit record.
User Key	Key for the User whose user snapshot is recorded in this entry.
From Date	Date the snapshot entry became effective.
To Date	Date the snapshot entry was no longer effective. In the case of the entry representing the current User Profile, the To data is set to NULL.
Delta	The XML representation of changes only.
User Profile Snapshot	The XML representation of the User Profile snapshot.
Source	The source of the entry. The username of the user responsible of the change in addition to the API used.
Signature	This column is for customers to sign the Snapshot for non-repudiation.

Trigger for Taking A Snapshot

When any data elements in the user profile snapshot change, Oracle Identity Manager to takes a new user profile snapshot. The following events trigger the creation of a new snapshot:

- Modification to the user record of any kind, for example, by recon, direct, adapter, and so on
- Group membership change for the user
- Changes in the policies that apply to the user
- Provisioning a resource to the user
- De-provisioning of a resource for the user
- Any provisioning related event for a provisioned resource:
  - Resource status change
  - Addition of provisioning tasks to the provisioning process
  - Updates to provisioning tasks in the provisioning process, for example, status changes, escalations, and so on
  - Creation of or updates to Process Form data

The Audit Engine

To prevent the volume of audit records from affecting performance, an audit engine performs complex data extraction, processing, and recording. The audit engine works between every step of processing in Oracle Identity Manager, instead of during transaction processing. When a trigger for a defined event fires, the audit engine takes a snapshot and performs offline processing to generate and record the snapshot.

The rest of section discusses the following topics:

- Audit Levels
- Using Post-Processors
- Creating Custom Post-Processors

Audit Levels

User profile auditing is enabled by default when you install the Audit and Compliance module, and the auditing level is set to `Resource Form`. After changing the auditing level, you should run the `GenerateSnapshot.sh` script on Linux or the `GenerateSnapshot.bat` script on Windows. This script examines all users in the Oracle Identity Manager database and generates new snapshots based on the auditing level.

Note: After changing the auditing level, be sure to run the `GenerateSnapshot` script before allowing users to access the system.

You can specify the level of detail for auditing, including the active triggers, the behavior of the audit engine, and the data captured in the audit snapshot.

You specify audit levels as a system configuration property in Oracle Identity Manager. The supported levels are:

- **Process Task:** Audits the entire user profile snapshot with the resource lifecycle process.
- **Resource Form:** Audits user record, group membership, provisioned resources, and any form data associated with the resource.

- **Resource:** Audits the user record, group membership, and provisioned resources.
- **Membership:** Only audits the user record and user group membership.
- **Core:** Only audits the user record.
- **None:** No audit is stored.

Note: Audit level specifications are case-sensitive.

Using Post-Processors

The AUD table stores the audit metadata XML. The audit engine uses the metadata to create the snapshot XML file and the XML file for changes to the snapshot. The metadata XML provides, among other things, information about the table that stores the snapshot, the XML file for changes, and post-processors. The post-processors process data after the audit engine generates the snapshot and change XML and stores it in the auditor table.

Types of Post-Processors

There are two types of post-processors, the internal auditor type and custom type. The internal auditor post-processors are defined in the auditor XML metadata. For instance, the user profile auditor has an internal post-processor that normalizes the XML filess into the reporting tables: UPA_USR, UPA_FIELDS, UPA_GRP_MEMBERSHIP, and UPA_RESOURCE. These tables are used by the reporting module to generate the appropriate reports.

Creating Custom Post-Processors

You create custom post-processors to extend the functionality or reporting of a given auditor. Custom post-processors are not defined in the XML metadata. You define them in the lookup tables in Oracle Identity Manager.

See also: Appendix A, "Sample Code for a Custom Post-Processor"

To create custom post-processors:

1. Create a new class that extends from the CustomAuditDataProcessor class and implements the processAuditData method.
2. Create a new Lookup Definition in the Oracle Identity Manager Design Console as follows:
   a. Call the code Audit.*AuditorName*.CustomProcessors, where *AuditorName* is the name of the auditor that the post-processor use.

      For example, for a user profile audit, the auditor name can be UserProfileAuditor.

   b. Select the **Lookup Type** option.

   c. Enter the group name related to the auditor, in this case, the UPA Processors Group.

   d. Add the Lookup Code Information.

      This is the fully qualified classpath to the class you created. The code key and Decode should be the classpath.

   e. Enter en for the Language and US for the Country settings.

3. After the class is created and the lookup information is set up, place the class in a .jar file in the *OIM_HOME*/JavaTasks directory.

Tables Used for Audits

User profile audits uses the following tables in the database:

- AUD: This table stores information on all the auditors that Oracle Identity Manager supports.

  Currently, only the UserProfileAudit entry is available.

- AUD_JMS: This table stores information about the changes made for an auditor.

  Currently, only user profile changes are stored. 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 re-issue messages if they are not processed.

- UPA: This table is the main table and stores all the snapshots and changes made to the user profiles.

The Oracle Identity Manager reporting module uses the following tables:

- UPA_USR: This table stores user profile information.
- UPA_FIELDS: This table stores user profile information in a vertical format.

  This table has more information than the UPA_USR table. For instance, UD fields are stored in this table as well as other fields that are not available in UPA_USR.

- UPA_GRP_MEMBERSHIP: This table contains group membership for all the users in the system.

  The information includes when a user was added and removed from a group. Currently only direct group membership is stored in this table.

- UPA_RESOURCE: The information in this table includes provisioned resources and changes in status for each of the resources.

  This table does not include any form table information.

Re-issue Audit Message Task

Oracle Identity Manager includes a scheduled task named Re-issue Audit Message Task. This task re-issues audit JMS messages that were not processed because of database connectivity problems. If problems are found when the JMS message is picked up for processing, the JMS system retries up to a configured number of times. If the message is not processed after the configured number of retries, it ends up in the *Dead Letter Queue*.

The actual data is not in the JMS system. All audit message data is stored in AUD_JMS. A corresponding JMS message with the AUD_JMS ID is sent. When the JMS message is picked up for processing, data from AUD_JMS is retrieved. You can re-issue the message by sending the ID from AUD_JMS back to the JMS system using the Re-issue Audit Message Task.

All messages about the same entity to be updated (for a user profile, the entity is the user key) reside in the AUD_JMS until they are resolved. After the Re-issue Audit Message Task runs, there should not be any AUD_JMS entries. If there are AUD_JMS entries, the message could be corrupted or the processor of the message cannot

understand it. Each audit for a single user is performed sequentially so that the audits are logged according to when the events happened.

By default, the Re-issue Audit Message Task runs daily, but you should configure a specific start date and time, preferably when the system is not too busy. This allows for enough time to process messages, especially if there are many of them. You should run this task regularly. Use Oracle Identity Manager Design Console configure a specific start date and time for the Re-issue Audit Message Task. You can also use Design Console to configure the Re-issue Audit Message Task as follows:

- Configure aud_jms entries to be processed according to an identifier
- Specify a specific auditor class.
- Configure a threshold, in number of hours, for the age that a message must be before it is submitted to the queue.

Release 9.0.3 introduces a new system flag, XL.SendAuditJMSMessage, configurable in the system properties section of the Design Console. The default value for the flag is True. When set to False, the audit engine creates an entry in aud_jms but does not send a JMS message. When set to False, you must run the ReissueAuditMessage scheduled task to process aud_jms entries.

Table 2–2 summarizes the attributes for the ReissueAuditMessage scheduled task that you can configure with Design Console:

Table 2–2 ReissueAuditMessage Attributes

Attribute	Values

Key: How the schedule task should process aud_jms entries.	The following are valid *keys*:
	`identifier`
	`all`
	Default: `all`

Value: The value corresponding to the key attribute.	The following are valid values: If the key is `identifier`, provide the identifier to be processed. Separate multiple values with a comma (","). If the key is `all`, this parameter does not have any effect.
`QueueDelay`	This parameter is a threshold, in number of hours, for the age that a message must be before it is submitted to the queue. The purpose of this parameter is to allow other tasks to be completed before processing the audit records for the tasks. Default: 5

Process Old Audit Messages Task

A new scheduled task named Process Old Audit Messages was added in Release 9.0.3. This scheduled task is reserved for future use and may be revised or deprecated in a future release. Do note enable or execute this task.

3

Oracle Identity Manager Reporting

Oracle Identity Manager includes a custom reporting engine that enables you to run predefined reports against the Oracle Identity Manager transactional database or a secondary database if one is configured. You can add new reports without editing any Java code, and you can obtain reporting data by invoking a stored procedure.

> **See Also:** *Oracle Identity Manager Administrative and User Console Guide* for a complete list of operational and historical reports that are installed with Oracle Identity Manager

This chapter discusses the following topics:

Reporting Features

The following are Oracle Identity Manager reporting features:

- Select and view reports from a predefined list in the Administrative and User Console.
- Use a delegated administration model to control the reports available to a user and the information included in those reports.
- Filter report information.
- View reports on-screen.
- Export reports to CSV files.
- Provide interactive reports.
- Run reports from a secondary database.

The following sections explain data storage for reporting at the data, XML, and API layers in Oracle Identity Manager.

Data Layer

You can change the database schema and add stored procedures in the data layer.

The REP and RPG tables support reporting. The REP table contains the following:

- A list of all reports in the system
- The report name
- The report code
- The report type
- The report description
- The name of the stored procedure for the report
- The name of the data source
- The maximum report size
- The number of filters to be displayed on the report page
- XML metadata for each report

The RPG table is a link table between the REP and UGP table. This table stores information about group permissions for reports.

Each report is associated with a stored procedure. To run a report, you run the associated stored procedure with relevant arguments. You cannot run a report based on a database query.

Since there can be many reports in the system, the stored procedure follows rules to enable the report to be invoked generically. See "How To Create A New Report" on page 3-3 for details.

Each stored procedure has a set of required generic parameters. These parameters provide standard information, for example, starting row, page size, filter columns, and so on. The stored procedure can also have any number of report-specific parameters. Each stored procedure returns two values: a result set representing a page of the entire report data, and a total count for the report data. The standard format for a stored procedure and the XML metadata enable you to add and run any report without changing any Java code.

XML Metadata

XML metadata for each report is stored in the REP table for the report. The metadata provides the following information for the report:

- Layout information
- Representation of all the input parameters and their association with the corresponding stored procedure parameters
- Support for user-defined parameters (operational reports only)
- Display information for each report input parameter, for example, a field label or field type (for example, TextField, LookupField, and so on)
- Location of each column on the report display page
- Display information for each report data column
- Columns to be included in the filter drop downs
- Clickable columns for interactive reports

For details on the metadata structure, see "How To Create A New Report" on page 3-3.

API Layer

The API layer provides all back-end reporting functionality. The back end is not tied to the front end. You can create custom user interfaces using the reporting APIs.

How To Create A New Report

The following are tasks for creating a new report:

- Writing the Stored Procedure
- Creating the Report XML Metadata
- Modifying Property Files for Translating Label Names, Report Names, and Report Descriptions
- Creating an REP Entry and Providing Access to the Report

Writing the Stored Procedure

Each report is based on a single stored procedure. The following are rules for the stored procedure:

- Use a stored procedure, not a user-defined function.
- A stored procedure returns two values: the report data result set and the total number of rows in the report.
- The report result set is paged.

  The result set that is returned when you run the stored procedure represents one page of the entire report data. The starting row and the size of the page is specified at the time of running the stored procedure.
- The stored procedure handles filter parameters and user defined input parameters.

Each stored procedure uses generic parameters, and each can use parameters specific to the stored procedure. These parameter types are described in the following sections.

Generic Parameters

Generic parameters are common to all the stored procedures. You specify generic parameters in a required sequence before any specific parameters.

There are twelve generic parameters. All twelve generic parameters are required, even if their values are null. The following are the generic parameters in the order that you must specify them:

1. Report Result Set (type=`cursor`, OUT): The result set that represents the report data.

   Note: For SQL Server, the return type is Integer (`int`) and not cursor. In SQL Server, data is returned in the last query. There is no actual return parameter. This parameter type meets the requirements of the stored procedure query.

2. User Key (type=`int`, IN): The key of the user who runs the report.

This user key ensures that only records are returned for which the user has read permissions.

3. Sort Columns (type=`varchar`, IN): A list of comma-delimited column names on which the report result set can be sorted.

   Reserved for future use.

4. Sort Order (type=`varchar`, IN): The sort order (ascending or descending) for the report result set.

   Reserved for future use.

5. Start Row (type=`int`, IN): The row number where the result set starts.

6. Page Size (type=`int`, IN): The size of the result set, or the number of entries in a single page in a multi-page report.

7. Do Count (type=`int`, IN): Can have values 0, 1, or 2.

   A value of 0 indicates that the result set is computed and returned, but the total number of rows in the entire report data is not computed. A value of 1 indicates that the result set and the total number of rows are computed. When the value is 2, only the total number of rows is computed and an empty result set is returned.

8. Total Rows (type=`int`, OUT): This is an OUT parameter that returns the total number of rows when the value of the Do Count variable is either 1 or 2.

   Since the report data is paged, the value of the total number of rows is not the size of the returned result set. It is the size of the entire report.

   For example, suppose that a stored procedure returns a list of all users. There are 200 users in the system, the start row=1, and the page size=50. For this stored procedure, the size of the result set is 50, but the value of the total rows OUT parameter is 200.

9. Filter Column Names (type=`varchar`, IN): This is a comma-delimited list of column names on which the report data can be filtered.

   A stored procedure has no way of knowing the alias to use for the listed columns. The stored procedure expects that the column names in this list are correctly qualified with the appropriate table aliases, if needed, for example:

   ```
   usr.usr_first_name,obj.obj_name
   ```

10. Filter Column Values (type=`varchar`, IN): This is a comma-separated list of column values that have a one-to-one correspondence with the column names listed in the previous parameter.

    If the previous parameter contains a comma-separated list with two column names, this parameter is a comma separated list of two values. You can use a wild card (%) character, for example, `Jo%,Laptop`.

11. User-Defined Column Names (type=`varchar`, IN): This is a comma-separated list of column names that represent user-defined columns on system forms.

    These names must be appropriately aliased if needed, for example:

    ```
    USR.USR_UDF_SSN
    ```

12. User-Defined Column Values (type=`varchar`, IN): This is a comma-delimited list of column values for user defined fields.

    These values have a one-to-one correspondence with the column names listed in the previous parameter. You can use the wild card (%) character, for example:

```
1234567890
```

Specific Parameters

Specific parameters are specific to each report. These parameters have a one-to-one correspondence with the report input parameters, except for the date range input parameter and user-defined parameters.

You must add specific parameters after the generic parameters. Each specific parameter represents one report input parameter on the Report Input page.

All specific parameters that are of the `varchar2` type support the wild card (%) character.

Other Stored Procedure Notes

Each time an error is encountered, an exception is thrown. The exception contains an error code. The calling Java code receives the error as a SQLException with the error code embedded in it. The stored procedure checks the code for errors based on the following rules:

- The value of Start Row cannot be 0 or null.
- The value of Page Size cannot be 0 or null.
- The value of User Key cannot be 0 or null.
- The value of Do Count can only be 0, 1 or 2.
- There is a one-to-one mapping between Filter Column Names and Filter Column Values.
- There is a one-to-one mapping between User-Defined Column Names and User-Defined Column Values.

Even if there is no data to return for a report, an empty result set is returned.

Example of a Stored Procedure Signature

Example 3–1 shows the signature for the User Resource Access report stored procedure for Oracle Database:

Example 3–1 Signature of the User Resource Access Stored Procedure for Oracle Database

```
PROCEDURE XL_SP_UserResourceAccess (
    csrresultset_inout            IN OUT   sys_refcursor,
    intuserkey_in                 IN       NUMBER,
    strsortcolumn_in              IN       VARCHAR2,
    strsortorder_in               IN       VARCHAR2,
    intstartrow_in                IN       NUMBER,
    intpagesize_in                IN       NUMBER,
    intdocount_in                 IN       NUMBER,
    inttotalrows_out              OUT      NUMBER,
    strfiltercolumnlist_in        IN       VARCHAR2,
    strfiltercolumnvaluelist_in   IN       VARCHAR2,
    strudfcolumnlist_in           IN       VARCHAR2,
    strudfcolumnvaluelist_in      IN       VARCHAR2,
    struserlogin_in               IN       VARCHAR2,
    strfirstname_in               IN       VARCHAR2,
    strmiddlename_in              IN       VARCHAR2,
    strlastname_in                IN       VARCHAR2,
    struseremail_in               IN       VARCHAR2,
```

```
    strorgname_in                IN       VARCHAR2,
    strusergroup_in              IN       VARCHAR2,
    strmgrfirstname_in           IN       VARCHAR2,
    strmgrlastname_in            IN       VARCHAR2,
    struserstatus_in             IN       VARCHAR2,
    struseremptype_in            IN       VARCHAR2
)
```

In Example 3–1, the first twelve parameters (upto `strudfcolumnvaluelist_in`) are generic parameters. The remaining parameters are specific parameters.

Example 3–2 illustrates the SQL Server signature for the User Resource Access stored procedure:

Example 3–2 SQL Server Signature for the User Resource Access Stored Procedure

```
CREATE PROCEDURE  XL_SP_UserResourceAccess
(@csrResultSet_inout                  INT OUTPUT,
@intUserKey_in                        INT,
@strSortColumn_in                     VARCHAR(4000),
@strSortOrder_in                      VARCHAR(4000),
@intStartRow_in                       INT,
@intPageSize_in                       INT,
@intDoCount_in                        INT,
@intTotalRows_inout                   INT OUTPUT,
@strFilterColumnList_in               VARCHAR(8000),
@strFilterColumnValueList_in          VARCHAR(8000),
@strudfcolumnlist_in                  VARCHAR(8000),
@strudfcolumnvaluelist_in             VARCHAR(8000),
@strUserLogin_in                      varchar(256),
@strFirstName_in                      varchar(80),
@strMiddleName_in                     varchar(80),
@strLastName_in                       varchar(80),
@strUserEmail_in                      varchar(256),
@strorgname_in                        varchar(256),
@strUserGroup_in                      varchar(30),
@strMgrFirstName_in                   varchar(80),
@strMgrLastName_in                    varchar(80),
@strUserStatus_in                     varchar(25),
@strUserEmptype_in                    varchar(255)
)
```

Creating the Report XML Metadata

After you create the stored procedure, you create the XML metadata for the report. All report-specific information goes into the metadata so that the report can be run and displayed correctly. The report metadata provides information such as attributes of the report-specific input parameters, the display properties of the report input parameters and the display properties of the report data, for example, report layout, display labels, and so on.

The root element of the metadata is `report`. This element provides the layout of the report. Three display layouts are supported: tabular layout, sectional layout, and sectional with report header.

The `report` element has two child elements: `StoredProcedure` and `ReturnColumns`.

The StoredProcedure Element

The `StoredProcedure` element provides information about the stored procedure-specific parameters and user defined fields. It consists of a single `InputParameters` element that contains multiple `InputParameter` elements.

Each specific stored procedure parameter corresponds to one input parameter on the Report Input page, except for the `DateRange` field, which is represented by two stored procedure parameters. Each input parameter is represented by one `InputParameter` element. The Report Input page can also contain user-defined fields from any system form. Each user-defined field on the Report Input page is also represented by one `InputParameter` element. The number of user-defined fields can change, but the number of stored procedure input parameters does not change each time. User-defined fields are represented by comma-delimited lists.

For example, suppose a report needs to support seven input parameters. Two parameters are user-defined fields, and five are specific parameters in the signature of the stored procedure (in addition to the twelve generic parameters). These are represented by five `InputParameter` tags. The two user-defined parameters are also represented by two `InputParameter` tags, but they do not have corresponding parameters in the stored procedure signature. Instead, they are passed as comma-delimited lists of column names and their values. Thus, there should be a total of seven `InputParameter` tags in the metadata.

If you want to support the `DateRange` input type, the Report Input page must be supported by two stored procedure parameters: one for the From date and the other for the To date.

The following are the attributes of the `InputParameter` element:

- `name` (required:Yes): The name of the input parameter.

  In case of non-user-defined input parameters, this value can be anything. However, for clarity, it should match the name of the corresponding stored procedure input parameter. For user-defined input parameters, this name is the column name of the user-defined column, prefixed by the required table alias, if needed.

- `parameterType` (required:Yes): The SQL data type of the corresponding stored procedure parameter.

  In case of user-defined input parameters, this value is `varchar`.

- `order` (required:Yes): The order of the report-specific input parameters.

  The ordering starts from 1. You must list the regular input parameters, and the user-defined input parameters later.

- `fieldType` (required:Yes): The type of the display field on the Report Input page.

  The supported input types are: `TextField`, `Date`, `DateRange`, `LookupField`, and `Combobox`.

- `fieldLabel` (required: Yes): The property value of the field label for this field.

  The property value is a value from the message resources property file (`xlWebAdmin.properties` in this case) which represents the actual label.

- `allowedValues` (required: No, unless `fieldType` is `Combobox`): Lookup codes associated with a combo box.

  The combo box input type supports only static values.

- `required` (required: No, default: false): If set to true, the user needs to provide a value for this field for the report to run.
- `udf` (required: No, default:false): For a a user defined field that is represented by the `InputParameter` element, this attribute must be present and have a value of true. Operational reports only.

If the attribute `fieldType` has a value of `LookupField`, there needs to be a child element under the `InputParameter` element called `ValidValues`. The reporting functionality supports three types of lookups: Lookup by code, lookup by method, and lookup by column. The following are the supported attributes of the `ValidValues` element:

- `lookupCode` (required: No): Must be present only if the lookup is by code.

  If it is, the value of this attribute is the lookup code.
- `lookupColumn` (required: No): Must be present only if the lookup is by column.

  If it is, the value of this attribute is the column code of the lookup column.
- `lookupMethod` (required: No): Must be present only if the lookup is by class or method.

  If it is, the value of this attribute is the name of method that provides the lookup values.
- `operationClass` (required: No): Must be present only if the lookupMethod attribute is present.

  The value of this attribute is the fully qualified name of the class that contains the lookup method.
- `displayColumns` (required: No): Must be present only if the `lookupMethod` attribute is present.

  It is a comma-delimited list of column codes which represent columns that is displayed in the lookup.
- `selectionColumn` (required: No): Must be present only if the `lookupMethod` or `lookupColumn` attributes are present.

  It represents the column code of the column, the value of which is saved in the database.

Examples of InputParameter tags

- Regular `TextField` input parameter:

```
<InputParameter name="strfirstname_in" parameterType="varchar2" order="2"
fieldType="TextField" fieldLabel="report.userResourceAccess.label.firstName"
required="false" />
```

- User-Defined input parameter:

```
<InputParameter name="USR.USR_UDF_SSN" parameterType="varchar2" order="11"
fieldType="TextField" fieldLabel="report.userResourceAccess.label.SSN"
required="false" udf="true" />
```

- Input parameter of type `Combobox`:

```
<InputParameter name="struserstatus_in" parameterType="varchar2" order="10"
fieldType="Combobox" allowedValues="Lookup.WebClient.Users.Status"
fieldLabel="report.userResourceAccess.label.userStatus" required="false" />
```

- Input parameter of type `LookupField` with `lookupCode`:

```
<InputParameter name="struseremptype_in" parameterType="varchar2" order="11"
fieldType="LookupField"
fieldLabel="report.userResourceAccess.label.employeeType" required="false" >
  <ValidValues lookupCode="Lookup.Users.Role"/>
</InputParameter>
```

- Input parameter of type `LookupField` with `lookupColumn`

```
<InputParameter name="struseremail_in" parameterType="varchar2" order="5"
fieldType="LookupField" fieldLabel="report.userResourceAccess.label.userEmail"
required="false" >
  <ValidValues lookupColumn="Users.Xellerate Type" selectionColumn="Lookup
Definition.Lookup Code Information.Decode" />
</InputParameter>
```

- Input parameter of type `LookupField` with `lookupMethod`

```
<InputParameter name="struserlogin_in" parameterType="varchar2" order="1"
fieldType="LookupField" fieldLabel="report.userResourceAccess.label.userLogin"
required="false" >
  <ValidValues lookupMethod="findUsersFiltered"
   operationClass="Thor.API.Operations.tcUserOperationsIntf"
   displayColumns="Users.User ID,Users.Last Name,Users.First Name"
   selectionColumn="Users.User ID"/>
</InputParameter>
```

If the attribute `fieldType` has a value of `DateRange`, you must include the two child element `InputStartDate` and `InputEndDate`. Each element must have the following attributes:

- `name` (required: Yes): The name of the parameter.

  This should match the name of the stored procedure parameter that represents this date parameter.

- `parameterType` (required: Yes): The SQL type of the stored procedure parameter that represents this date parameter.
- `order` (required: Yes): The order of the stored procedure parameter
- `defaultValue` (required: No, default: 01/01/1900 and 12/31/2049): The default value to provide if the user does not enter a date in the start or end date fields.
- `format` (required: No, default: `reports.generic.message.internalDateFormat`): The format of the default date.

ReturnColumns tag

The `ReturnColumns` tag represents the list of all the columns that are being returned by the result set. This tag contains multiple `ReturnColumn` tags, each of which represents one column in the returned result set. The attributes of the `ReturnColumn` tag provides information that is useful for displaying the report data.

The following are the attributes of the `ReturnColumns` tag:

- `name` (required: Yes): This name represents the column code of the result set column that is represented by this particular tag.

  If the column code is not available, it can be the alias or the column name.

- `label`: (required: Yes): This provides the property value of the column header/label for this column.

  The property value is a value from the message resources property file (`xlWebAdmin.properties` in this case), which represents the actual label.

- `position` (required: Yes): This attribute can have three values: `Table`, `Sectional Header`, or `Report Header`.

  This attribute specifies the location of each column. Each column can reside in the table (for any layout), the section header (for Sectional Layout and Sectional with Report Header Layout), or the report header (in case of Sectional with Report Header layout).

- `filterColumn` (required: No, default: false): This attribute specifies whether the column is a filter column.

  If the value is true, then the column name is included in the filter drop down lists at the top of the Report Display page.

- `filterColumnName` (required: No, unless `filterColumn` is present): This attribute represents the actual name of the column prefixed by the table alias, if needed.

- `filterType` (required: No, unless `filterColumn` is false): The type of the filter display field on the Report result page. The only supported input type is `Combobox`.

- `filterLookupKey` (required: No, unless `filterColumn` is false): Represents a lookup code associated with a combo box that is shown for a filter display field on the Report result page.

- `clickable` (required: No, default: false): This attribute specifies whether the column value is a link.

  This attribute provides support for action-ability of reports.

You can configure interactive reports in the report module. In an interactive report, you define column values to be links that take the user to a page in or outside of the Adminitrative and User Console when he or she clicks the column value. To configure the links, you add information to the metadata so that the user is taken to the appropriate page. The links can have dynamic or static locations. Clicking on a link opens a new browser window that shares the same browser session but is otherwise self-sufficient.

To configure the links, set the `clickable` attribute to `true`, and configure the `ReturnColumn` element and its two child elements `Link` and `RequestParameters`, as follows:

- `Link` element: Configure the single attribute `href` for this element.

  This attribute specifies the base URL of the destination page. An absolute URL begins with `http`, for example, `http://www.xyz.com`. The destination page for an absolute URL is outside the Adminitrative and User Console. A relative URL, for example, `searchResources.do`, leads to a destination page in the Adminitrative and User Console.

- `RequestParameters` element: Configure multiple `RequestParameter` elements for this element.

  Each `RequestParameter` element represents one request parameter that must be submitted for the destination page to display properly. The `name` attribute specifies the name of the request parameter. If the request parameter is static, that

is, it does not depend on any other value in the result set, you configure its value using the `value` attribute. If the value of the request parameter is dynamic, that is, it depends on another column of the result set, for example, the Resource Key, you specify the value using the `column` attribute. The `column` attribute contains the column code for the column whose value is to be returned.

Creating an REP Entry and Providing Access to the Report

Once the report metadata is complete, update the `REP` and `RPG` tables to make the report available.

Creating a New Entry in the REP Table

The `REP` table contains a list of all the reports in the system. Defining a new report requires creating a new row in the `REP` table representing this report. Apart from the common columns, for example, Create Date, Created By, Row Version, and so on, you need to specify the columns that are populated. The values in the parentheses are examples for User Resource Access report.

- `REP_NAME`: This column contains the name of the report that is displayed to the user.

  This name is unique in the `REP` table. (User Resource Access)

- `REP_CODE`: A unique code for the report. (`UserResourceAccess`)
- `REP_DESCRIPTION`: A description for the report that is displayed to the user. (Resource access rights for selected users).
- `REP_SP_NAME`: The name of the stored procedure that provides the data for the report. (`XL_SP_UserResourceAccess`)
- `REP_XML_META`: This column is a `clob` that contains the entire metadata for the report defined in the previous section.
- `REP_TYPE`: The type of the report. This can have two values: `Operational` or `Historical`. (`Operational`)
- `REP_DATASOURCE`: The name of the data source against which the report is run.

  This can have two values: `Default` or `Reporting`. Usually, the Operational reports are run against the Default database while the Historical reports are run against the reporting database. (Default)

- `REP_MAX_REPORT_SIZE`: This represents the maximum number of records a report can return.

  Different reports will return different amount of data for each record. In order to keep the response time for a report acceptable, a maximum number of records that a report can return is enforced for each report. (5000)

- `REP_FILTER_COUNT`: The number of filter drop downs on the Report Display page for this report (3).

The following is an example of the insert statement that populates the `REP` table with the User Resource Access report data for an Oracle Database:

```
INSERT INTO REP (REP_KEY, REP_CODE, REP_TYPE, REP_NAME, REP_DESCRIPTION,
                 REP_DATASOURCE, REP_SP_NAME, REP_MAX_REP_SIZE, REP_FILTER_COUNT,
                 REP_DATA_LEVEL, REP_CREATE, REP_CREATEBY, REP_UPDATE,
                 REP_UPDATEBY, REP_ROWVER)
VALUES (rep_seq.nextval, 'UserResourceAccess', 'Operational', 'User Resource
Access','Resource access rights for selected users', 'Default',
```

```
'XL_SP_UserResourceAccess', 5000, 3, 1, SYSDATE, <System Administrator User Key>,
SYSDATE, <System Administrator User Key>, HEXTORAW('0000000000000000'));
```

The following is the example of the INSERT statement that populates the REP table with the User Resource Access report data in SQL Server:

```
INSERT INTO REP (REP_CODE, REP_TYPE, REP_NAME, REP_DESCRIPTION, REP_DATASOURCE,
                 REP_SP_NAME, REP_MAX_REP_SIZE, REP_FILTER_COUNT, REP_DATA_LEVEL,
                 REP_CREATE, REP_CREATEBY, REP_UPDATE, REP_UPDATEBY, REP_ROWVER)
VALUES ('UserResourceAccess', 'Operational', 'User Resource Access',
       'Resource access rights for selected users', 'Default',
'XL_SP_UserResourceAccess',
       5000, 3, 1, GETDATE(), <System Administrator User Key>, GETDATE(), <System
Administrator User Key>, 0x0);
```

Loading XML Metadata

After you create an entry in the REP table for the report, you must load the XML metadata for the report into the REP table's REP_XML_META column. See "Creating the Report XML Metadata" on page 3-6 for information on creating the metadata.

Providing a User Group with Access to a Report

The following procedure describes providing access to a report.

> **See also:** Oracle Identity Manager Administrative and User Console Guide

To provide access to the new report to a particular user group:

1. Search for the user group using the Manage User application.
2. Navigate to the detail page for the user group.
3. Click Allowed Reports link in the additional details drop down.

   The Reports page under Group Detail appears.
4. Click the **Assign Reports** button.

   The Assign Reports page appears in the Reports section under Group Detail. The report you have just created is listed on this page.
5. To provide access to the report, assign it to at least one group.

Modifying Property Files for Translating Label Names, Report Names, and Report Descriptions

This section describes how to modify property files for translating label names, report names, and report descriptions. It contains the following topics:

- Adding Properties for Translating Label Names
- Adding Properties for Translating Report Names and Descriptions

Adding Properties for Translating Label Names

After you write the XML metadata, you must introduce the new field label properties in the metadata file in the properties files. You add the properties to the files that correspond to the locales that you need to support. The languages that are supported in Release 9.0.3 and their associated property files are as follows:

- English (xlWebAdmin_en_US.properties)

- French (xlWebAdmin_fr.properties)
- German (xlWebAdmin_de.properties)
- Chinese, Simplified (xlWebAdmin_zh_CN.properties)
- Chinese, Traditional (xlWebAdmin_zh_TW.properties)
- Italian (xlWebAdmin_it.properties)
- Japanese (xlWebAdmin_ja.properites)
- Korean (xlWebAdmin_ko.properties)
- Portuegese, Brazilian (xlWebAdmin_pt_BR.properties)
- Spanish (xlWebAdmin_es.properties)

When Oracle Identity Manager encounters an unsupported locale, it refers to the xlWebAdmin.properties file. You add to these files the properties included as `fieldLabel` attributes of `InputParameter` tags and label attributes of `ReturnColumn` tags. Instead of providing the actual name of the input field labels or return column labels, you specify the property name, which is then looked up from the respective properties file. This makes it simpler for internationalization.

The following examples illustrate how to set the `fieldLabel` attributes of the `InputParameter` and `ReturnColumn` tags.

Example 1

```
<InputParameter name="struserlogin_in" parameterType="varchar2" order="1"
fieldType="TextField" fieldLabel="report.userResourceAccess.label.userLogin"
required="false" />
```

This `InputParameter` tag is from `UserResourceAccess.xml` metadata file for the User Resource Access report. The `fieldLabel` attribute of this tag has the value of `report.UserResourceAccess.label.userLogin`. The corresponding entry for this property in the `xlWebAdmin_en_US.properties` file is:

```
report.UserResourceAccess.label.userLogin=Userid
```

Example 2

```
<ReturnColumn name="Users.First Name"
label="report.userResourceAccess.label.firstName"
 position="SectionHeader" filterColumn="true"
 filterColumnName="usr.usr_first_name" />
```

This `ReturnColumn` tag is from `UserResourceAccess.xml` metadata file for the User Resource Access report. The `label` attribute of this tag has the value of `report.userResourceAccess.label.firstName`. The corresponding entry for this property in the `xlWebAdmin_en_US.properties` file is:

```
report.userResourceAccess.label.firstName=First Name
```

> **Note:** If the actual label names change (as a result of bug fixes, for example), the property names need not be changed.

You perform the following steps to edit the xlWebAdmin_en_US.properties file:

1. Extract the xlWebApp.war file in the `%` *XL_HOME*`%/`webapp directory to a temporary directory.

2. Open in a text editor the xlWebAdmin_en_US.properties file in the temporary directory where you extracted the xlWebApp.war file.
3. Add the appropriate properties as `fieldLabel` attributes of `InputParameter` tags and label attributes of `ReturnColumn` tags.
4. Recreate the xlWebApp.war file and copy it to the `%` *XL_HOME*%/webapp directory.

Adding Properties for Translating Report Names and Descriptions

After you create an entry in the `REP` table, you must add translation properties for the report name and description to the properties files. This procedure is necessary to localize the English entries in the database and must be performed even if you only need to provide support for the English locale.

To add properties for translating report names and descriptions:

1. Extract the xlWebApp.war file in the % *XL_HOME*%/webapp directory to a temporary directory.
2. Open in a text editor the xlDefaultAdmin.properties file from the temporary directory where you extracted the xlWebApp.war file.
3. Locate in the xlDefaultAdmin.properties file the `global.resultSet.Reports.Report~Name` property, which identifies the names of all reports in the system. Each report assigned to the property is separated by a pipe character (`|`). Append a pipe character and the name of the new report to the property value. Ensure that there are no spaces between the pipe character and the report name, and that there are no trailing spaces following the report name.
4. Locate in the xlDefaultAdmin.properties file the `global.resultSet.Reports.Report~Description` property, which contains descriptions of all reports in the system. Each report assigned to the property is separated by a pipe character (`|`). Append a pipe character and the description of the new report to the property value. Ensure that there are no spaces between the pipe character and the description, and that there are no trailing spaces following the description.
5. Open in a text editor the properties file representing the locale for which you want to translate report names and descriptions.
6. Add a global.resultSet.Reports.Report~Name.*ReportName* property, which identifies the report name and its localized value. The *ReportName* portion of the property name represents the name of the report. Spaces in the property name are represented by tildes (~). You assign a localized value to the property. For example, you add the following property to the xlWebAdmin_en_US.properties file to represent a report named Group Information:

   ```
   global.resultSet.Reports.Report~Name.Group~Information=Group Information
   ```

7. Add a global.resultSet.Reports.Report~Description.*Report~Description* property, which contains a report description along with and its localized value. The *Report~Description* portion of the property name represents the default report descriptions. Spaces in the report description are represented by tildes (~). You assign a localized value to the property. For example, you add the following property to the xlWebAdmin_en_US.properties file for a description of the Group Information:

   ```
   global.resultSet.Reports.Report~Description.Description~of~the~Group~Informatio
   ```

```
n~report=Description of the Group Information report
```

8. Add global.ReportName.Lookup.*Report-Name* and global.ReportDesc.Lookup.*Report-Description* lookup properties for each new report. The *Report-Name* and *Report-Description* portions of each property name represent the default report name and description, respectively. Spaces in the report description are represented by hyphens (-). You assign localized values to each property. For example, you add the following properties to the xlWebAdmin_en_US.properties file for the Group Information report:

   ```
   global.ReportName.Lookup.Group-Information=Group Information
   global.ReportDesc.Lookup.Description~of~the~Group~Information~report=Descriptio
   n of the Group Information report
   ```

9. Recreate the xlWebApp.war file and copy it to the `%` *XL_HOME*%/webapp directory.

Working with Third-Party Reporting Tools

Third-party reporting tools can run reports against Oracle Identity Manager by using the provided stored procedures. You do not need to understand the data model or write queries for predefined reports. You can use any reporting tool for custom stored procedures and custom reports.

Information about the snapshot and changes are stored in XML form in the `UPA` table. A third-party XML reporting tool can generate reports from this table. If XML is not a desired format, the reporting tool can use the reporting tables related to the user profile audit feature to retrieve data: `UPA_USR`, `UPA_FIELDS`, `UPA_GRP_MEMBERSHIP`, and `UPA_RESOURCE`.

4

Secondary Datasource Reporting

You can configure Oracle Identity Manager to use one database for current transactional data and a secondary database for historical data. The secondary database eases the load on the transactional database.

You can use different data sources for the secondary database. The following sections describe how to configure Oracle Identity Manager and your application server to use a secondary data source.

This chapter discusses the following topics:

Writing User Profile Audits to a Secondary Datasource

User profile audit data can increase in size quickly. Oracle recommends that you use a secondary database to store this information. The following system property enables reading and writing to this database directly:

`XL.UserProfileAuditInSecondaryDS`.

By default, the `XL.UserProfileAuditInSecondaryDS` property is set to `false`. If this property is set to `true`, the system reads and writes all user profile data directly to and from the secondary database.

If you configure a secondary database, all historical reports are automatically configured to run against it.

The user profile audit interacts directly with the secondary database. You must replicate other tables from the transactional database because the report needs them for access control and filtering of the report. You can disable these tables and constraints for ease of data backup, restore, or replication.

Table 4–1 lists the tables and contraints.

Table 4–1 Tables and Constraints Used in Historical Reports

Table Name	Foreign Key Constraint Name	Referenced Table Name	Referenced Column Name
AAD	FK_AAD_FK_AAD_AC_ACT	ACT	ACT_KEY
	FK_AAD_FK_AAD_UG_UGP	UGP	UGP_KEY
ACT	FK_ACT_ACT	ACT	PARENT_KEY
	FK_ACT_SRP	SRP	SRP_KEY
GPG	FK_GPG_UGP	UGP	UGP_KEY
	FK_GPG_UGP_KEY_UGP	UGP	GPG_UGP_KEY
OUG	FK_OUG_OBJ	OBJ	OBJ_KEY
	FK_OUG_UGP	UGP	UGP_KEY
POL			
PTY			
REQ	FK_REQ_ORC	ORC	ORC_KEY
	FK_REQ_OST	OST	OST_KEY
	FK_REQ_USR	USR	USR_KEY
UGP			
USG	FK_USG_RUL	RUL	RUL_KEY
	FK_USG_UGP	UGP	UGP_KEY
	FK_USG_USR	USR	USR_KEY
USR	FK_USR_ACT	ACT	ACT_KEY

Steps to Set Up a Secondary Data Source

To set up a secondary database:

1. Create the secondary database.

   You can back up and restore the transactional database under a different database name, or you can replicate the transactional database.

2. Set up the application server to use the secondary database.

   See the following sections for details.

3. Set the system property `XL.UserProfileAuditInSecondaryDS` to `True`.

   This stores user profile audit data in the secondary database.

4. Configure daily replication of the data so that the tables in the secondary database listed in Table 4–1 are updated from the primary database. After the secondary database is functional, do not replicate the entire primary database in the secondary database or any audit data that has been stored in the secondary database will be deleted.

   Or, set up either a full restore or replication.

5. Make sure all stored procedures are replicated correctly in the secondary database.

6. Define a connection URL as follows:

   - For Oracle Database:

```
jdbc:oracle:thin:@<IP of database>:<SID>
```

- For SQL Server:

```
jdbc:Microsoft:sqlserver://<IP of
database>:<Port>;DatabaseName=<SID>;SelectMethod=Cursor
```

Using JBoss with a Secondary Data Source

To create a new data source on JBoss, a new file called xlreportds-service.xml is created by the setup in the deployment directory. This file creates an alias to the transactional database using the `java:jdbc/xlXAReportingDS` setting.

To point to a secondary database on JBoss:

1. Open the xell-ds.xml file in an editor.
2. Add the following to xell-ds.xml as a second `xa-datasource` tag for the Oracle database:

```
<xa-datasource>
<jndi-name>jdbc/xlXAReportingDS</jndi-name>
<track-connection-by-tx>true</track-connection-by-tx>
<isSameRM-override-value>false</isSameRM-override-value>
<xa-datasource-class>oracle.jdbc.xa.client.OracleXADataSource </xa-datasource-
class>
<xa-datasource-property name="URL">jdbc:oracle:thin:@<IP of database system>:
1521:XELL </xa-datasource-property>
<xa-datasource-property name="User">sysadm</xa-datasource-property>
<xa-datasource-property name="Password">sysadm</xa-datasource-property>
<exception-sorter-class-name> org.jboss.resource.adapter.jdbc.vendor.
OracleExceptionSorter </exception-sorter-class-name>
<no-tx-separate-pools/>
<valid-connection-checker-class-name> org.jboss.resource.adapter.jdbc.vendor.
OracleValidConnectionChecker </valid-connection-checker-class-name>
</xa-datasource>
```

For SQL Server, the secondary database tag is as follows:

```
<xa-datasource>
  <jndi-name>jdbc/xlXADS</jndi-name>
  <track-connection-by-tx>true</track-connection-by-tx>
  <xa-datasource-class>
    com.microsoft.jdbcx.sqlserver.SQLServerDataSource</xa-datasource-class>
  <xa-datasource-property name="ServerName"><IP of database system>
  </xa-datasource-property>
  <xa-datasource-property name="DatabaseName">XELL</xa-datasource-property>
  <xa-datasource-property name="SelectMethod">cursor</xa-datasource-property>
  <xa-datasource-property name="PortNumber">1433</xa-datasource-property>
  <user-name>sysadm</user-name>
  <password>sysadm</password>
  <check-valid-connection-sql>
    select 1 from USR where 1=2
  </check-valid-connection-sql>
</xa-datasource>
```

Note that the class names for Oracle Database and SQL Server vary as indicated:

- Oracle:

```
oracle.jdbc.xa.client.OracleXADataSource
```

- SQL Server:

```
com.microsoft.jdbcx.sqlserver.SQLServerDataSource
```

3. Change the database name, user name, and password to connect to the database you set up as the secondary database.
4. Delete the xlreportds-service.xml file.
5. Restart the JBoss server.

> **Note:** Do not add the `xa-datasource` block in this section or point the `jdbc/xlXAReportingDS` to the transactional database because it causes errors. To point to the same transactional database, keep the xlreportds-service.xml file as is.

Cluster Configuration for JBoss

In a standalone setup, the xell-ds.xml and xlreportds-service.xml files are in the *JBOSS_HOME*\server\default\deploy\ directory.

In a clustered setup, the file xell-ds.xml is in the *JBOSS_HOME*\server\all\farm\ directory, and xlreportds-service.xml in the JBO*SS*_HOME\server\all\deploy\ directory.

To configure a cluster for JBoss:

1. Copy the changes to the xell-ds.xml file to all computers in the cluster.
2. Restart the JBoss servers on all computers in the cluster.

Using WebLogic with a Secondary Data Source

Before changing the data source that Oracle Identity Manager uses for reporting, create a new data source in WebLogic. Follow the WebLogic manuals to set up a new data source.

To configure WebLogic with a secondary data source using the Oracle database:

1. Log in to the WebLogic administrative console.
2. Navigate to JDBC Connection Pools
3. Create a Connection Pool with the following credentials:
   - **Name**: `xlXAReportConnectionPool`
   - **URL**: `jdbc:oracle:thin:@<`*`database IP address`*`>:<`*`port no`*`>:<`*`SID`*`>`
   - **Class Name**: `oracle.jdbc.xa.client.OracleXADataSource`
   - **Username**: *<secondary database user name>*
   - **Password**: *<secondary database password>*
4. Create a secondary data source and deploy it on the server.

   Navigate to JDBC Data Sources on the WebLogic administrative console and create a data source with the following credentials:
   - **JNDI name**: `jdbc/xlXAReportingDS`
   - **Pool Name**: `xlXAReportConnectionPool`
5. Edit the file weblogic.profile *XL_HOME*/xellerate/Profiles/ to point to the new data source.

Add the JNDI name to weblogic.profile as follows:

```
datasource.report=jdbc/xlXAReportingDS
```

6. Run the patch command (`patch_weblogic`) for the changes to take effect.

Cluster Configuration for WebLogic

To configure a cluster for WebLogic, you must deploy the secondary data source on all members of the cluster.

Using WebSphere with a Secondary Data Source

Before changing the data source used by Oracle Identity Manager for reporting, you must create a new data source in WebSphere. See the WebSphere manuals for information on setting up a new data source.

To configure WebSphere with a secondary data source using the Oracle database:

1. Log in to the WebSphere administrator console.
2. Create a new data source with the following details:
   - **Name**: *<XAReportingDataSource>*
   - **JNDI name**: `jdbc/xlXAReportingDS`
3. Define the connection URL as follows:

   `jdbc:oracle:thin:@<`*IP of database*`>:<`*port_number*`>:<`*SID*`>`

   For example: `jdbc:oracle:thin:@192.168.161.134:1521:xeltest`
4. Use the following J2C authentication data values:
   - **Alias**: *<secondary user alias>*
   - **User**: *<secondary user>*
   - **Password**: *<secondary user password>*
   - **Description**: *<Descriptive text for the data>*
5. Select the component-managed authentication aliases for `XAReportingDatasource` with the following values:
   - **Component-managed authentication alias**: *<J2C Authentication Data Entries>*
   - **Container-managed authentication alias**: *<J2C Authentication Data Entries>*
6. Save and synchronize changes among all nodes.
7. Open the file websphere.profile, and add the JNDI information that points to the new data source in the *XL_HOME*/xellerate/Profiles/ directory.

   Comment out the existing data source entry for `xlXADS` and add the information for `xlXAReportingDS` as follows:

```
# Reporting data source
#datasource.report=jdbc/xlXADS
 datasource.report=jdbc/xlXAReportingDS
```

8. Set the following Java Client System property to true:

```
XL.UserProfileAuditInSecondaryDS=True
```

9. Run `patch_websphere.cmd` or `patch_websphere.sh` as applicable from the *XL_HOME*\xellerate\setup directory.

Cluster Configuration for WebSphere

To configure a cluster for WebSphere:

1. Modify each websphere.profile file on all nodes participating in the cluster.
2. Run the `patch_websphere.cmd` or `patch_websphere.sh` as applicable from the *XL_HOME*\xellerate\setup directory from the network deployment manager (NDM) node.
3. Stop and restart all nodes and servers.

Using OC4J with a Secondary Data Source

Before changing the data source used by Oracle Identity Manager for reporting, you must create a new data source in OC4J.

To configure OC4J with a secondary data source using the Oracle database:

1. Log in to the Oracle Enterprise Manager 10*g* Application Server Control Console for the OC4J instance.
2. Click **Administration** to display the list of administration tasks you can perform on the selected OC4J instance.
3. If necessary, expand the Services section of the table by clicking the expand icon or by clicking **Expand All**.
4. Click the task icon in the table's JDBC Resources row.

   The Application Server Control Console displays the JDBC Resources page, which lists the data sources and connection pools currently available in the OC4J instance.
5. Click the **xlXAReportingConnectionPool** link under the connection pools section.
6. Define the connection URL as follows:

   `jdbc:oracle:thin:@<IP of database>:<port_number>:<SID>`

   For example: `jdbc:oracle:thin:@192.168.161.134:1521:xeltest`
7. Enter the secondary database user name in the **User Name** field.
8. Select the **Use Cleartext Password** option and provide the secondary database user password.
9. Click **Apply**.
10. Restart the OC4J instance.

A

Sample Code for a Custom Post-Processor

The sample post-processor in this section gets the group entitlements from the Active Directory integration. The Active Directory integration uses a child table to store the group membership.

Create a table to store the information you need in the reporting database using the following SQL scripts.

```
CREATE SEQUENCE UPA_UD_ADUSRC_SEQ
INCREMENT BY 1
START WITH 1
CACHE 20

/*==============================================================*/
/* Table: UPA_UD_ADUSRC
*/
/*==============================================================*/
CREATE TABLE UPA_UD_ADUSRC (
  UPA_UD_ADUSRC_KEYNUMBER(19)       NOT NULL,
  UPA_RESOURCE_KEYNUMBER(19)       NOT NULL,
  OIU_KEYNUMBER(19)       NOT NULL,
  UD_ADUSRC_GROUPNAMEVARCHAR2(256)    NOT NULL,
  STATUSVARCHAR2(7),
  UPA_UD_ADUSRC_EFF_FROM_DATE TIMESTAMP        NOT NULL,
  UPA_UD_ADUSRC_EFF_TO_DATETIMESTAMP,
  CREATE_DATETIMESTAMP        NOT NULL,
  UPDATE_DATETIMESTAMP        NOT NULL,
  CONSTRAINT PK_UPA_UD_ADUSRC PRIMARY KEY (UPA_UD_ADUSRC_KEY)
)

COMMENT ON TABLE UPA_UD_ADUSRC IS
'Stores AD group entitlements'

CREATE INDEX IDX_UPA_UD_ADUSRC_EFF_FROM_DT ON UPA_UD_ADUSRC (
   UPA_UD_ADUSRC_EFF_FROM_DATE ASC
)
```

Use the following code for the custom post-processor:

```
package sample.audit.processor;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
```

```
import java.sql.Statement;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.sql.Types;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

import com.thortech.xl.audit.auditdataprocessors.CustomAuditDataProcessor;
import com.thortech.xl.audit.engine.AuditData;
import com.thortech.xl.audit.exceptions.AuditDataProcessingFailedException;
import com.thortech.xl.util.logging.LoggerMessages;

public class ADUserGroupMembershipProcessor extends CustomAuditDataProcessor {

    private static final String CHANGE_TAG = "Change";
    private static final String ATTRIBUTE_TAG = "Attribute";
    private static final String NAME_ATTRIBUTE = "name";
    private static final String CHANGE_LOCATION_ATTRIBUTE = "where";
    private static final String ACTION_ATTRIBUTE = "action";

    private static final String CHILD_DATA_PREFIX = "/Data";
    private static final String RESOURCE_DATA_PREFIX =
"/ProcessData/Children/Child";
    private static final String RESOURCE_PROFILE_PREFIX =
"/UserProfileSnapshot/ResourceProfile/ResourceInstance";

    private static final String AD_RESOURCE_NAME = "AD User";

    /*private static final String[] UPA_UD_ADUSRC_COLUMNS =
        {"UPA_UD_ADUSRC_KEY", "OIU_KEY", "UD_ADUSRC_KEY", "UD_ADUSRC_GROUPNAME",
         "UPA_UD_ADUSRC_EFF_FROM_DATE", "UPA_UD_ADUSRC_EFF_TO_DATE",
         "CREATE_DATE", "UPDATE_DATE"};*/

    public void processAuditData(Connection operationalDB,
            Connection reportingDB, List auditDataList, Timestamp auditEpoch)
            throws AuditDataProcessingFailedException {
        for (Iterator iter = auditDataList.iterator(); iter.hasNext();) {
            AuditData auditData = (AuditData) iter.next();
            // Retrieve data from AuditData value object
            //String auditeeID = auditData.getAuditeeID();
            List changeElements = getChangeElements(auditData.getChanges());
            // Retrieve AD User Group Membership related changes
            List ADUserGrpMembershipChangeElements =
                getADUserGroupMembershipChangeElements(changeElements);
            // Process change elements
            for (Iterator iterator =
ADUserGrpMembershipChangeElements.iterator();iterator.hasNext();) {
                Element changeElement = (Element) iterator.next();
                // Retrieve the resource instance key (OIU_KEY) from the XPath
expression
                long resourceInstanceKey = getResourceInstanceKey(changeElement);
                // Get the object name for this resource instance key
                String resName = getResourceName(auditData.getUpdatedSnapshot(),
resourceInstanceKey);
```

```
                if(resName == null || !resName.equals(AD_RESOURCE_NAME))
                continue;// this is not the AD User resource so, skip it and check
the next one...
                // Retrieve the child table key (UD_ADUSRC_KEY)
                long UDADUSRCKey = getUDADUSRCKey(changeElement);
                // Retrieve the current record, if present
                HashMap ADUserGroupMembershipProfile =
                    readADUserGroupMembershipData(reportingDB,
resourceInstanceKey, UDADUSRCKey);
                // Reset the default columns
                ADUserGroupMembershipProfile.put("UPA_UD_ADUSRC_KEY",null);
                ADUserGroupMembershipProfile.put("OIU_KEY",null);
                ADUserGroupMembershipProfile.put("UD_ADUSRC_KEY",null);
                // Apply the changes
                String action = changeElement.getAttribute(ACTION_ATTRIBUTE);
                if (action.equalsIgnoreCase("Delete")) {
                    ADUserGroupMembershipProfile.put("STATUS","DELETE");
                } else {

ADUserGroupMembershipProfile.put("STATUS",action.toUpperCase());
                    Element groupNameElement =

getFirstChildElementByName(changeElement,ATTRIBUTE_TAG,NAME_ATTRIBUTE,"UD_ADUSRC_G
ROUPNAME");
                    Attribute attrDetails = getAttributeDetails(groupNameElement);

ADUserGroupMembershipProfile.put("UD_ADUSRC_GROUPNAME",attrDetails.getNewValue());
                }
                // Set values for the default columns
                ADUserGroupMembershipProfile.put("OIU_KEY",new
Long(resourceInstanceKey));
                ADUserGroupMembershipProfile.put("UD_ADUSRC_KEY",new
Long(UDADUSRCKey));
                ADUserGroupMembershipProfile.put("UPA_UD_ADUSRC_EFF_FROM_DATE",
auditEpoch);
                ADUserGroupMembershipProfile.put("UPA_UD_ADUSRC_EFF_TO_DATE",
null);
                ADUserGroupMembershipProfile.put("CREATE_DATE", new
Timestamp(System.currentTimeMillis()));
                ADUserGroupMembershipProfile.put("UPDATE_DATE", new
Timestamp(System.currentTimeMillis()));
                // Update existing active record if present
                updateActiveADUserGroupMembershipProfile(reportingDB,
resourceInstanceKey, UDADUSRCKey, auditEpoch);
                // Insert new record
                insertNewADUserGroupMembershipProfile(reportingDB,
ADUserGroupMembershipProfile);
            }
        }
    }

    private long insertNewADUserGroupMembershipProfile(Connection reportingDB,
            HashMap ADUserGroupMembershipProfile)
            throws AuditDataProcessingFailedException {
        long key = 0;
        try {
            String insertSQL =

generateNewADUserGroupMembershipInsertSQL(reportingDB,ADUserGroupMembershipProfile
);
```

```
            key = executeInsert(reportingDB, insertSQL,
ADUserGroupMembershipProfile);
        } catch (SQLException e) {
            String errMsg = "Unable to insert new AD User Group Membership
Profile";
            throw new AuditDataProcessingFailedException(errMsg,e);
        }
        return key;
    }

    private String generateNewADUserGroupMembershipInsertSQL(Connection
reportingDB,
            HashMap ADUserGroupMembershipProfile) throws SQLException {
        String valuesPlaceHolder = "";
        String columnNames = "";
        String dbType = reportingDB.getMetaData().getDatabaseProductName();

        if (dbType.startsWith("Oracle")) {
            valuesPlaceHolder = "?, ";
            columnNames = "UPA_UD_ADUSRC_KEY, ";
        }

        for (Iterator iter = ADUserGroupMembershipProfile.keySet().iterator();
iter.hasNext();) {
            String columnName = (String) iter.next();
            Object columnValue = ADUserGroupMembershipProfile.get(columnName);
            if (!columnName.equals("UPA_UD_ADUSRC_KEY") && columnValue != null) {
                valuesPlaceHolder += "?, ";
                columnNames += columnName + ", ";
            }
        }

        // Trim the place holder variable and column names variable
        valuesPlaceHolder =
(valuesPlaceHolder.trim()).substring(0,valuesPlaceHolder.length()-2);
        columnNames = (columnNames.trim()).substring(0,columnNames.length()-2);

        String insertSQL = "INSERT INTO UPA_UD_ADUSRC (" + columnNames + ") " +
                           "VALUES (" + valuesPlaceHolder + ")";

        return insertSQL;
    }

    private void updateActiveADUserGroupMembershipProfile(Connection reportingDB,
            long resourceInstanceKey, long UDADUSRCKey, Timestamp auditEpoch)
            throws AuditDataProcessingFailedException {
        String updateSQL = "UPDATE UPA_UD_ADUSRC " +
                             "SET UPA_UD_ADUSRC_EFF_TO_DATE=?" +
                           "WHERE OIU_KEY=?" +
                             "AND UD_ADUSRC_KEY=?" +
                             "AND UPA_UD_ADUSRC_EFF_TO_DATE is null";

        try {
            PreparedStatement pstmt = reportingDB.prepareStatement(updateSQL);
            pstmt.setTimestamp(1,auditEpoch);
            pstmt.setLong(2,resourceInstanceKey);
            pstmt.setLong(3,UDADUSRCKey);
            pstmt.executeUpdate();
        } catch (SQLException e) {
            String errMsg = "Failed to update active AD user group membership
```

```
profile. " +
                            "Data in UPA_UD_ADUSRC could be inconsistent";
            if (dbLogger.isDebugEnabled())
                dbLogger.debug(errMsg,e);
            throw new AuditDataProcessingFailedException(errMsg, e);
        }

    }

    /**
     *
     * @param operationalDB
     * @param resourceInstanceKey
     * @param UDADUSRCKey
     * @return
     * @throws AuditDataProcessingFailedException
     */
    private HashMap readADUserGroupMembershipData(Connection reportingDB,
            long resourceInstanceKey, long UDADUSRCKey)
            throws AuditDataProcessingFailedException {
        String query = "SELECT * " +
                        "FROM UPA_UD_ADUSRC " +
                       "WHERE OIU_KEY=" + resourceInstanceKey + " " +
                         "AND UD_ADUSRC_KEY=" + UDADUSRCKey + " " +
                         "AND UPA_UD_ADUSRC_EFF_TO_DATE is null";
        try {
            return transformADUserGroupMembershipProfile(executeQuery(reportingDB,
query));
        } catch (SQLException e) {
            throw new AuditDataProcessingFailedException("Unable to read data from
" +
                    "JDBC resultset", e);
        } catch (Exception e) {
            String errMsg = "AD User Group Membership information stored in " +
                            "UPA_UD_ADUSRC is inconsistent";
            if (dbLogger.isDebugEnabled())
                dbLogger.debug(errMsg, e);
            throw new AuditDataProcessingFailedException(errMsg, e);
        }
    }

    /**
     *
     * @param result
     * @return
     * @throws Exception
     */
    private HashMap transformADUserGroupMembershipProfile(ResultSet result)
            throws Exception {
        HashMap ADUserGroupMembershipProfile = new HashMap();
        ResultSetMetaData metadata = result.getMetaData();

        // Move the cursor to the beginning of the resultset
        if (result.next()) {
            //
            for (int i = 0; i < metadata.getColumnCount(); i++) {
                Object columnValue = null;
                String columnName = metadata.getColumnName(i+1);
                int columnType = metadata.getColumnType(i+1);
                switch (columnType) {
```

```
                    case Types.INTEGER:
                        columnValue = new Long(result.getLong(i+1));
                        break;
                    case Types.VARCHAR:
                    case Types.CHAR:
                    case Types.LONGVARCHAR:
                        columnValue = result.getString(i+1);
                        break;
                    case Types.TIMESTAMP:
                        columnValue = result.getTimestamp(i+1);
                        break;
                    default:
                        columnValue = null;
                }
                if (result.wasNull()) {
                    columnValue = null;
                }
                ADUserGroupMembershipProfile.put(columnName, columnValue);
            }
            // Check if more than one record was returned. If so throw an
exception
            if (result.next()) {
                String errMsg = "More than one active record found for AD group" +
                                result.getString("UD_ADUSRC_GROUPNAME");
                throw new Exception(errMsg);
            }
        }

        return ADUserGroupMembershipProfile;
    }

    /**
     *
     * @param changeElement
     * @return
     */
    private long getResourceInstanceKey(Element changeElement) {
        long resourceInstanceKey = 0;
        String changeLocation =
changeElement.getAttribute(CHANGE_LOCATION_ATTRIBUTE);
        int keyStartPosition =
changeLocation.indexOf(RESOURCE_PROFILE_PREFIX)+RESOURCE_PROFILE_PREFIX.length()+7
;
        int keyEndPosition =
keyStartPosition+changeLocation.substring(keyStartPosition).indexOf("'")-1;
        resourceInstanceKey =
Long.parseLong(changeLocation.substring(keyStartPosition,keyEndPosition+1));
        return resourceInstanceKey;
    }

    /**
     *
     * @param changeElement
     * @return
     */
    private String getResourceName(Document snapshot, long resourceInstanceKey) {

    Element parentElement = snapshot.getDocumentElement();
    NodeList childNodes = parentElement.getChildNodes();
```

```
        for (int i = 0; i < childNodes.getLength(); i++) {
            Node childNode = childNodes.item(i);

            if ((childNode.getNodeType() == Node.ELEMENT_NODE) &&
                    childNode.getNodeName().equals("ResourceProfile")) {
            NodeList resourceProfileNodeList = childNode.getChildNodes();
            for (int j = 0; j < resourceProfileNodeList.getLength(); j++) {
            Node resNode = childNodes.item(j);
                    if ((resNode.getNodeType() == Node.ELEMENT_NODE) &&
                    resNode.getNodeName().equals("ResourceInstance")) {
                    Element resourceInstanceElement = (Element)resNode;
                    String key = resourceInstanceElement.getAttribute("key");
                    if(key != null && Long.parseLong(key) == resourceInstanceKey)
                    {
                    Element name =
getFirstChildElementByName(resourceInstanceElement, ATTRIBUTE_TAG, NAME_ATTRIBUTE,
"Objects.Name");
                    return name.getFirstChild().getNodeValue();
                    }
                    }
}

            }
        }
        return null;
    }

    /**
     *
     * @param changeElement
     * @return
     */
    private long getUDADUSRCKey(Element changeElement) {
        long UDADUSRCKey = 0;
        String changeXPath =
changeElement.getAttribute(CHANGE_LOCATION_ATTRIBUTE);
        String processDataXPath =
changeXPath.substring(changeXPath.indexOf(RESOURCE_DATA_PREFIX));
        String childDataXPath =
processDataXPath.substring(processDataXPath.indexOf(CHILD_DATA_PREFIX));
        int keyStartPosition = CHILD_DATA_PREFIX.length()+7;
        int keyEndPosition =
keyStartPosition+childDataXPath.substring(keyStartPosition).indexOf("'")-1;
        UDADUSRCKey = Long.parseLong(childDataXPath.substring(keyStartPosition,
keyEndPosition+1));
        return UDADUSRCKey;
    }

    /**
     *
     * @param changeElements
     * @return
     */
    private List getADUserGroupMembershipChangeElements(List changeElements) {
        List ADUserGrpMembershipChangeElements = new ArrayList();

        for (Iterator iter = changeElements.iterator(); iter.hasNext();) {
            Element change = (Element) iter.next();
            if (isChildrenData(change.getAttribute(CHANGE_LOCATION_ATTRIBUTE)))
                ADUserGrpMembershipChangeElements.add(change);
```

```
        }

        return ADUserGrpMembershipChangeElements;
    }

    /**
     *
     * @param attribute
     * @return
     */
    private boolean isChildrenData(String changeLocation) {
        if (changeLocation.startsWith(RESOURCE_PROFILE_PREFIX) &&
                changeLocation.indexOf(RESOURCE_DATA_PREFIX,
RESOURCE_PROFILE_PREFIX.length()) > 0)
            return true;
        else
            return false;
    }

    /**
     *
     * @param connection
     * @param query
     * @return
     */
    protected ResultSet executeQuery(Connection connection, String query) {
        ResultSet result = null;
        try {
            Statement stmt = connection.createStatement();
            if (stmt.execute(query)) {
                result = stmt.getResultSet();
            }
        } catch (SQLException e) {
            if (dbLogger.isDebugEnabled()) {
                dbLogger.debug(LoggerMessages.getMessage("DBQueryExecutionError",
query),e);
            }
        }
        return result;
    }

    /**
     *
     * @param connection
     * @param userGroupMembershipProfile
     * @param updateSQL
     */
    protected long executeInsert(Connection connection, String insertSQL,
            HashMap ADUserGroupMembershipProfile) throws SQLException {
        PreparedStatement insertStmt = connection.prepareStatement(insertSQL);
        String dbType = connection.getMetaData().getDatabaseProductName();

        long newKey = 0;
        int columnIndex = 1;
        //
        // Get new key for Oracle and set it into the prepared stmt
        if (dbType.startsWith("Oracle")) {
            ResultSet nextValRS = executeQuery(connection, "select
UPA_UD_ADUSRC_SEQ.nextval from dual");
            long nextVal = nextValRS.getLong(1);
```

```
            insertStmt.setLong(columnIndex++,nextVal);
            newKey = nextVal;
        }

        // Set column names and values for other columns in UPA_UD_ADUSRC
        for (Iterator iter = ADUserGroupMembershipProfile.keySet().iterator();
iter.hasNext();) {
            String columnName = (String) iter.next();
            Object columnValue = ADUserGroupMembershipProfile.get(columnName);
            if (!columnName.equals("UPA_UD_ADUSRC_KEY") && columnValue != null) {
                if (columnValue.getClass().getName().endsWith("Long")) {

insertStmt.setLong(columnIndex++,((Long)columnValue).longValue());
                } else if (columnValue.getClass().getName().endsWith("String")) {
                    insertStmt.setString(columnIndex++,(String)columnValue);
                } else if (columnValue.getClass().getName().endsWith("Timestamp"))
{
                    insertStmt.setTimestamp(columnIndex++,(Timestamp)columnValue);
                }
            }
        }

        insertStmt.executeUpdate();

        if (dbType.startsWith("Microsoft SQL Server")) {
            ResultSet nextValRS = executeQuery(connection, "select @@identity");
            long nextVal = nextValRS.getLong(1);
            newKey = nextVal;
        }
        return newKey;
    }
}
```

Index

P

R

S

T

U

W

X