Oracle® Identity Manager

Audit Report Developer's Guide

Release 9.0

B28760-01

May 2006

ORACLE®

Oracle Identity Manager Audit Report Developer's Guide, Release 9.0

B28760-01

Primary Authors: Craig West, Don Biassotti, Deepa Aswani

Contributors: Mario Lim, Semyon Shulman

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 and Upgrade Guide for JBoss*
- *Oracle Identity Manager Installation and Upgrade Guide for WebLogic*
- *Oracle Identity Manager Installation and Upgrade Guide for WebSphere*
- *Oracle Identity Manager Administrative and User Console Guide*
- *Oracle Identity Manager Design Console Guide*
- *Oracle Identity Manager Administrative and User Console Customization Guide*
- *Oracle Identity Manager Tools Reference Guide*
- *Oracle Identity Manager Best Practices Guide*

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 includes audit and compliance reporting functionality that captures and archives entity and transaction data. This archived data is used for IT-centric process and forensic auditing, and compliance monitoring.

The archived data indicates which users have access to what information, the purpose of this access, and the means by which the information is made available. The entire lifecycle of the historical data can be recorded, including capture, transport, storage, retrieve, and removal. Data security is maintained at every part of the data lifecycle.

The historical data, reporting engine and interface are key infrastructure components used by other Oracle Identity Manager features and solutions. User profile audit, reports, and attestation are auditing features of Oracle Identity Manager with the audit and compliance modules. This guide includes details related to the User Profile Audit and Reporting. See the *Oracle Identity Manager Administrative and User Console Guide* for Attestation details.

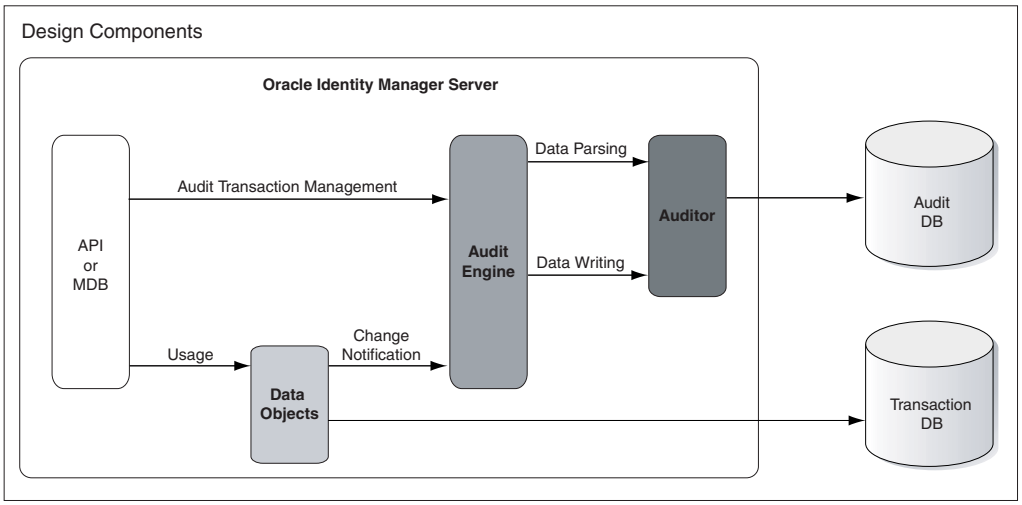
This chapter introduces you to the following concepts related to auditing using Oracle Identity Manager:

- Auditing Design Components
- User Profile Auditing
- Standard and Customized Reports
- Secondary Data Source Reporting

Auditing Design Components

Figure 1–1 shows the design components of the auditing process.

Figure 1–1 Design Components of the Auditing Process



Any action taken in the Oracle Identity Manager system translates into an application programming interface (API) call, or an MDB picking up a message to process some action.

Multiple changes could originate from this one action. All associated changes are tied together into one *Audit Transaction*. Each API method that can modify data objects calls the `startTransaction` method on the Audit Engine at the beginning of the API and the `endTransaction` method at the end of the method call, defining the Audit Transaction boundaries. The Audit Engine generates a transaction ID that is used to identify all changes made in that transaction.

User Profile Auditing

Oracle Identity Manager provides auditing and historical archiving of a user profile. The system takes a snapshot of a user profile and stores that snapshot to an audit table in the database. The system then updates the snapshot each time the user data changes.

Standard and Customized Reports

Oracle Identity Manager includes standard reports for displaying archived data in addition to the capability for users to create customized reports to suit their specific needs.

Secondary Data Source Reporting

Oracle Identity Manager comes configured to issue reports from a secondary data source. Out of the box, the software uses the primary data source for reporting, `jdbc/xlDS`. To prevent overloading the database that is used for transaction data to create reports, a new data source can be setup just for reporting. To use a secondary database, you need to configure replication of data or a scheduled backup and restore between the transactional data and the reporting database.

2

Oracle Identity Manager User Profile Auditing

User profile audit stores information about changes in the user profile, user membership, resource provisioning, access policies, and resource forms.

This chapter discusses user profile auditing in the following sections:

Data Collected

By default, User Profile Audit is enabled (when installing with Audit and Compliance module) and the auditing level is set to `Resource From`. The auditing level is configurable and specifies the minimum level required for attestation on form data.

The `XL.UserProfileAuditDataCollection` keyword in System Properties in the Oracle Identity Manager Administrative and User Console determines the audit level. See Audit Levels in this chapter for more detailed information. This section covers the following topics:

Capture and Archiving of User Profile Data

The system takes a snapshot of a user profile and stores that snapshot in an audit table in the database. Oracle Identity Manager updates the snapshot any time the user data changes.

The following outline describes what constitutes a saved user profile. It also provides information about 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

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)

The snapshot for a user profile contains all the previous data for a particular user.

XML Representation of the User Profile Snapshot

The snapshot and changes are saved as XML strings. Representing them in XML makes for easy readability and understanding of what events happened to update the snapshot. The following sections include examples of the XML in the snapshot and changes field in the UPA table.

Snapshot XML File

The snapshot XML describes all the attributes for a given user. All the attributes are related to the user profile. The top most tag is UserProfileSnapshot. Within this tag, there are a key and version. These store user key and version information for each XML entry. Each of the subsequent tags store information about the user profile:

- UserInfo: Information about the user profile itself
- GroupMembership: Information about the group membership
- PolicyProfile: Information about what policy allowed which resource provisioning
- ResourceProfile: Information about all the provisioned resources
  - ResourceInstance: Information on each of the resources provisioned to the user
  - ProcessData: Information about the data stored in the UDFs

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

```
  <?xml version="1.0" encoding="ISO-8859-1" ?>
- <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">2006-01-05 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">2006-01-05 17:11:56.868</Attribute>
    <Attribute encrypted="true" name="Users.Password"
               password="true">8YxO3YSKDXJLmcsKeZhUSw == </Attribute>
    <Attribute name="Users.Creation Date">2006-01-05 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">2006-01-05 17:12:30.299
      </Attribute>
      <Attribute name="Groups-Users.Update Date">2006-01-05 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">2006-01-05
                 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">2006-01-05
                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>
```

Snapshot Changes XML File

The snapshot changes XML describes all the changes that affect user attributes for a given transaction. All the attributes are related to the user profile. The top-most tag is `Changes`. Under this tag, we find all the changes made for a particular transaction. Each `Change` tag refers to a set of changes that affect the snapshot XML. The `where` attribute locates the position of the change and the Audit Engine makes the changes to the earlier snapshot XML by applying the changes.

The following extract is from an XML file containing snapshot changes:

```
 <?xml version="1.0" encoding="ISO-8859-1" ?>
- <Changes>
-   <Change action="insert" order="1"
    where="/UserProfileSnapshot/ResourceProfile/ResourceInstance[@key='74']">
-     <Attribute name="Users-Object Instance For User.Creation Date">
        <OldValue />
        <NewValue>2006-01-05 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

Every time a snapshot of a user profile is taken, it is stored in a User Profile Audit, or 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

This section defines the events that trigger Oracle Identity Manager to take a user profile snapshot. As mentioned earlier, Oracle Identity Manager takes a new snapshot whenever any of the data elements that constitute the user profile snapshot change. Thus, the events that trigger the creation of a new snapshot are:

- Modification to the user record (whatever the source: recon, direct, adapter, and so forth.)
- Group membership change for the user
- Changes in the policies that apply to the user
- Provisioning of 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 (status changes, escalations, and so on.)
- Creation of or updates to Process Form data

Audit Engine

To prevent the volume of audit records to become a drain on the performance of the system, a conceptual Audit Engine performs complex data extraction, processing, and recording activity between every step of processing in Oracle Identity Manager, instead of the transactional processing. When the defined Event Triggers fire, they notify the audit engine to take a snapshot. The audit engine then performs the necessary processing offline to generate and record the audit snapshot.

This section includes the following topics:

Audit Levels

When you install the Audit and Compliance module, User Profile Audit is enabled by default and the auditing level is set to Resource From. After upgrading or changing the auditing level to a different value, 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.

Tuning controls allow you to specify the level of detail of the auditing. These controls define the active triggers, and the working of the audit engine, and the data captured in the audit snapshot.

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

- Process Task: Audits the entire user profile snapshot together with the Resource Lifecycle Process.
- Resource Form: Audits user record, group membership, resource provisioned, and any form data associated to the resource.
- Resource: Audits the user record, group membership, and resource provisioning.
- 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, which is used by the audit engine to create the snapshot and changes XML. Apart from a lot of other information, this metadata XML provides information on the table to store the snapshot and changes XML and post-processors. Post-processors are used to process data after the audit engine generates the snapshot and changes 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 XMLs 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

Custom post-processors are not included with the auditor itself. These are created by customers to extend the functionality or reporting of a given auditor. Custom post-processors are not defined in the XML metadata but in the lookup tables in Oracle Identity Manager.

To create custom post-processors, do the following:

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 will be using. For instance, in the case of User Profile audit, the auditor name would 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, which is the fully qualified classpath to the class you created. The code key and Decode should be the classpath. Type 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 *XL_HOME*/JavaTasks directory.

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

Tables Used for Audits

User profile audit uses the following tables in the database:

- AUD: This table stores information on all the auditors supported by Oracle Identity Manager. Currently, only the UserProfileAudit entry is available. More auditors are planned in future releases to audit other type of entities (for example, User Groups, Resources, and so on).

- AUD_JMS: This table stores the 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. With this table, Oracle Identity Manager can control the order of the changes when multiple changes are made to the same entity (in this case User). Also, 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 following tables are used for reporting using the Oracle Identity Manager reporting module:

- UPA_USR: This table stores user profile only.
- 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 the group. Currently only direct group membership is stored in this table.
- UPA_RESOURCE: The information in this table includes the provisioned resources and the status change of 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 called Re-issue Audit Message Task that re-issues Audit JMS messages that were not processed because of database connectivity problems. All audit message data is stored in AUD_JMS and 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.

If any problems are found when the JMS message is picked up for processing, the JMS system will retry up to the configured number of retry times. If the message is not processed after retrying the configured number of retry times, it will end up in the *Dead Letter Queue*. Since the actual data is not in the JMS system, but is in another table, 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.

Once the Re-issue Audit Message Task runs, there should not be any AUD_JMS entries. If there are any AUD_JMS entries, the message itself could be corrupted or the processor of the message cannot understand it. All messages concerning the same entity (in the case of User Profile, the entity is the user key) to be updated will reside in the AUD_JMS until they are resolved. Each of the audits for a single user are performed sequentially so that the audits are logged in order of when the event happened.

You should enable the Re-issue Audit Message Task and set it to run regularly. By default, the Re-issue Audit Message Task runs daily, but you should configure a specific start date and time. The date and time specified should preferably be around times when the system is not too busy. This allows for enough time to process messages, especially if there are too many of them to process.

3

Oracle Identity Manager Reporting

Oracle Identity Manager includes a custom reporting engine so users can run predefined reports against the Oracle Identity Manager transactional database or a secondary database, if one is configured. The reporting module is flexible so that adding new reports is simple, without editing any Java code. You may obtain all the reporting data by invoking a stored procedure. Oracle Identity Manager comes with the following out-of-the-box operational and historical reports:

- Who Has What
- Resource Access List
- User Resource Access History
- User Profile History
- Resource Access List History

This chapter includes the following topics:

Reporting Features

You can use the Oracle Identity Manager reporting features in the following ways:

- Retrieve report data based on a predefined list of standard reports, available as stored procedures.
- Select and view reports from a predefined list in the Administrative and User Console.
- Use a delegated administration model that controls the reports available to a user, and the information included in those reports.
- Filter report information.
- View reports paged on-screen.
- Export reports as 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

The data layer is where you can make changes to the database schema and add stored procedures. To support the reporting functionality, two new tables have been introduced. They are `REP` and `RPG`.

The `REP` table contains a list of all the reports present in the system. This table includes the name, report code, type, description, stored procedure name, data source name, maximum supported report size, and the number of filters to be displayed on the report page. It also contains the XML meta data for each report.

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

Each report is associated with a stored procedure. To run a report, you need to run the associated stored procedure with the relevant arguments. There is no facility to run a report based on a database query.

Since there can be many reports in the system, the stored procedure follows certain rules so that the report can be generically invoked. These rules are listed in detail in the How To Create A New Report section of this chapter.

Each stored procedure must have some generic input parameters that provide standard information to the stored procedure (like start row, page size, filter columns, and so forth.). Apart from these generic parameters, the stored procedure can 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 of the report data. With a standard format of the stored procedure, and the provided XML meta data, any report can be added and run without changing any Java code.

XML Meta Data

The XML meta data for each report is stored in the `REP` table for that report. The meta data provides the following information for each report:

- Layout information for each report
- Representation of all the report input parameters and their association with the corresponding stored procedure parameters
- Support for user-defined parameters
- Display information for each report input parameter, such as display field label, field type (whether `TextField`, `LookupField`, and so forth)
- 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 a detailed description of the meta data structure, please refer to the How To Create A New Report section of this chapter.

API Layer

The API layer is written so that all the back-end reporting functionality is available. The reporting back end is not tied to the reporting front end. Using the reporting APIs, the administrator can write custom UIs.

How To Create A New Report

The process of creating a new report can be divided into the following tasks:

- Writing the Stored Procedure
- Creating the Report XML Meta Data
- Modifying the xlWebAdmin.properties File
- Creating REP Entries and Providing Access to the Report

Writing the Stored Procedure

Each report is based on a single stored procedure. This stored procedure provides all the report data when invoked. The reporting functionality follows a set of rules for the stored procedures. These rules are:

- It is a stored procedure and not a user-defined function.
- Each 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 each time the stored procedure is run represents one page of the entire report data. The starting row and the size of this page is specified at the time of running the stored procedure.
- The stored procedure handles filter parameters and user defined input parameters.

The parameters of each stored procedure are of two types: Generic Parameters and Specific Parameters. These are described in the following subsections.

Generic Parameters

Generic parameters are common to all the stored procedures. There are 12 generic parameters, a list of which follows. The order of the generic parameters is important because the reporting functionality expects these to be in a predefined order. The generic parameters are specified before any specific parameters.

All twelve generic parameters are necessary, even if most of the values are null. The generic parameters *in the order of specification* are:

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

> **Note:** In the case of SQL Server, the return type is Integer (`int`) and not cursor. The data that needs to be returned in SQL Server is returned in the last query, therefore, there is no actual return parameter. This parameter type is used to meet 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 is required so that only those 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 should 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 starting row number from 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. When the value is 0, the result set is computed and returned, but the total number of rows of the entire report data is not computed. When the value is 1, the result set and the total number of rows is computed. When the value is 2, only the total number of rows is computed and the result set is not computed and returned. Instead, 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 result set being returned, but the size of the entire report.

   For example, if a stored procedure returns a list of all users, and there are 200 users in the system, and start row=1, and page size=50, then the size of the result set returned 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. Since the stored procedure has no way of knowing which alias to use for the listed columns, it expects that the column names in this list are correctly qualified with the appropriate table aliases, if needed. An example of this is: "usr.usr_first_name,obj_name".
10. Filter Column Values (type=varchar, IN): This is a comma separated list of column values corresponding to the column names listed in the previous parameter. There is a one-to-one correspondence between the column names and column values. So if the previous parameter has a comma separated list having 2 column names, then this parameter is a comma separated list of 2 values. Also, the values support the wild card (%) character. An example of this is: "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 is appropriately aliased if needed. An example of this is: "USR.USR_UDF_SSN".
12. User-Defined Column Values (type=varchar, IN): This is a comma-delimited list of column values of user defined fields corresponding to the column names listed in the previous parameter. There is a one-to-one correspondence between the column names and the column values. Also, the values support the wild card (%) character. An example of this is: "1234567890".

Specific Parameters

The specific parameters are specified *after* the generic parameters. Each specific parameter represents one report input parameter on the Report Input page.

Specific parameters are, as indicated by the name, 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. 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 with an error code embedded in it so that the calling Java code can receive 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 in the values of the 'filter column names' and 'filter column values'.
- There is a one-to-one mapping in the values of the '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

The following is the signature of the Who Has What report stored procedure for Oracle Database:

```
PROCEDURE XL_SP_WhoHasWhat (
    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 this example, the first 12 parameters (upto `strudfcolumnvaluelist_in`) are all generic parameters and the remaining are specific parameters.

Here is the SQL Server Signature for the Who Has What stored procedure:

```
CREATE PROCEDURE  XL_SP_WhoHasWhat
```

```
(@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 Meta Data

After creating the stored procedure, create the XML meta data for the report. Since the report functionality is generic, all the report-specific information goes into the meta data, so that the report can be run and displayed correctly. The report meta data provides information such as attributes of the report specific input parameters, the display properties of the report input parameters and also the display properties of the report data (such as report layout information, display labels, and so on).

The root tag of the meta data is the `report` tag. This tag provides the layout of the report. The following three display layouts are supported: Tabular layout, Sectional layout, and Sectional with Report Header layout.

The `report` tag has two child tags: `StoredProcedure` and `ReturnColumns`.

The StoredProcedure tag

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

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 such input parameter is represented by one `InputParameter` tag. Apart from this, the Report Input page can also contain user-defined fields from any system form. In such a case, each user-defined field on the Report Input page is also represented by one `InputParameter` tag. The number of user-defined fields can change, but the number of stored procedure input parameters does not change each time. Hence, the user-defined fields are represented by comma-delimited lists.

For example, if the report needs to support 7 input parameters and 2 are user-defined fields, then there are 5 specific parameters in the signature of the stored procedure (apart from the 12 generic parameters). These are represented by 5 `InputParameter` tags. The 2 user-defined parameters are also represented by 2 `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 7 `InputParameter` tags in the meta data.

If the `DateRange` input type has to be supported, then that single input type on the Report Input page is 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` tag:

- `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 matches 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 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. It is required that the regular input parameters be listed first, and the user-defined input parameters come 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`): A list of comma separated values that is populated in the drop down of the 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): If the field represented by the `InputParameter` tag is a user defined field, then this attribute must be present and have a value of true.

If the attribute `fieldType` has a value of `LookupField`, then there needs to be a child tag under the `InputParameter` tag 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` tag:

- `lookupCode` (required: No): Must be present only if the lookup is by code. If it is, then 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, then 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, then 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.whoHasWhat.label.firstName"
required="false" />
```

- User-Defined input parameter:

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

- Input parameter of type Combobox:

```
<InputParameter name="struserstatus_in" parameterType="varchar2" order="10"
fieldType="Combobox" allowedValues=",Active,Disabled,Deleted"
fieldLabel="report.whoHasWhat.label.userStatus" required="false" />
```

- Input parameter of type LookupField with lookupCode:

```
<InputParameter name="struseremptype_in" parameterType="varchar2" order="11"
fieldType="LookupField" fieldLabel="report.whoHasWhat.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.whoHasWhat.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.whoHasWhat.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, then include the two child tags InputStartDate and InputEndDate, each of which having the following attributes:

- name (required: Yes): The name of the parameter. As a standard, this matches 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 be provided if the user does not enter any 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 itself.
- 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 either in the table (in case of any layout) or the sectional header (in case of Sectional Layout and Sectional with Report Header Layout) or 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.
- clickable (required: No, default: false): This attribute specifies whether the column value is a link. This attribute provides support for action-ability of reports.

The report module allows interactive reports, that is, selected column values can be links. In order to make the values links, extra information needs to be included in the meta data so that the user is taken to the appropriate page when the column value is clicked. Depending on how the links are configured, the user can either be taken to a page inside the Adminitrative and User Console, or to a separate page outside of the Adminitrative and User Console. The links can either have dynamic or static locations. Clicking on any link opens a new browser window that shares the same browser session but is otherwise self-sufficient.

In order to achieve this functionality, the ReturnColumn tag contains two child tags Link and RequestParameters if the clickable attribute has a value of true.

The Link tag has a single attribute called href which provides the base URL of the destination page. If the URL provided is absolute (begins with http, like

http://www.xyz.com) then the destination is a page outside the Adminitrative and User Console. If the URL provided is relative (for example, searchResources.do), then the destination page is an Adminitrative and User Console page.

The RequestParameters tag has multiple RequestParameter tags. Each RequestParameter tag represents one request parameter that needs to be submitted for the destination page to display properly. The name attribute specifies the name of the request parameter. The value of the request parameter can be specified in two ways. If the value of the request parameter is static (that is, it does not depend on any other value in the result set) then the value attribute provides that value. 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), then the value is specified by the column attribute. The column attribute contains the column code of the column whose value is to be returned.

Modifying the xlWebAdmin.properties File

After the XML metadata has been written, the new field label properties used in the metadata file must be introduced in the xlWebAdmin.properties file. These include all the properties included as the fieldLabel attributes of InputParameter tags and the label attributes of ReturnColumn tag. 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 xlWebAdmin.properties file. This makes it simpler for internationalization.

To edit the xlWebAdmin.properties file, extract the xlWebApp.war file present in the % *XL_HOME*%/webapp directory into a temporary directory. Navigate to the WEB-INF/classes directory and access the xlWebAdmin.properties file. Edit the xlWebAdmin.properties file in place. After the edits are done, re-create the xlWebApp.war file and put it in the % *XL_HOME*%/webapp directory. Then run the patch, depending on which application server is being used to host Oracle Identity Manager.

Example: xlWebAdmin.properties File Entries

The following examples illustrate the usage of the InputParameter and ReturnColumn tags in the xlWebAdmin.properties file.

Example 1

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

This InputParameter tag is from WhoHasWhat.xml metadata file for the Who Has What report. The fieldLabel attribute of this tag has the value of report.whoHasWhat.label.userLogin. The corresponding entry for this property in the xlWebAdmin.properties file is:

```
report.whoHasWhat.label.userLogin=Userid
```

Example 2

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

This ReturnColumn tag is from WhoHasWhat.xml metadata file for the Who Has What report. The label attribute of this tag has the value of

report.whoHasWhat.label.firstName. The corresponding entry for this property in the xlWebAdmin.properties file is:

```
report.whoHasWhat.label.firstName
```

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

Creating REP Entries and Providing Access to the Report

Once the report meta data is complete, update the REP and RPG tables to make the report available.

Updating 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 (like 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 Who Has What 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. (Who Has What)
- REP_CODE: A unique code for the report. (WhoHasWhat)
- 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_WhoHasWhat)
- REP_XML_META: This column is a clob that contains the entire meta data 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 Who Has What 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, 'WhoHasWhat', 'Operational', 'Who Has What','Resource
access rights for selected users', 'Default', 'XL_SP_WhoHasWhat', 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 Who Has What 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 ('WhoHasWhat', 'Operational', 'Who Has What',
       'Resource access rights for selected users', 'Default', 'XL_SP_WhoHasWhat',
       5000, 3, 1, GETDATE(), <System Administrator User Key>, GETDATE(), <System
Administrator User Key>, 0x0);
```

Providing Access to the Report

To provide access to the new report to a particular user group, search for the user group using the Manage User functionality and navigate to the detail page for that user group. Click Allowed Reports link in the additional details drop down, which will display the Reports page under Group Detail. On this page click Assign Reports button to navigate to the Assign Reports page in the Reports section under Group Detal. On this page you will see the name of the new report you have just created.

Assign this report to the user group from this page.

Note: *Oracle Identity Manager Administrative and User Console Guide*

Working with Third-Party Reporting Tools

Third-party reporting tools can run reports against Oracle Identity Manager by using the stored procedures provided. No understanding of the data model or writing queries for predefined reports is required. Any reporting tool can be used for custom stored procedures and custom reports.

Information about the snapshot and changes are stored in XML form in the UPA table and a third-party XML reporting tool can generate reports from this table. However, if XML is not a desired format, the reporting tool can rely on 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

Oracle Identity Manager can be configured to use two databases, one for transactional data (current data) and another for historical data. The historical reporting database is called the secondary database through this guide. This secondary database eases the load on the transactional database.

Different data sources can be used as the secondary database. The following example describes how to configure Oracle Identity Manager to use JBoss, WebLogic, and WebSphere servers to configure a the secondary data source.

This chapter covers the following topics:

Writing User Profile Audit to Secondary Datasource

Since user profile audit data can increase in size considerably and at a fast rate, it is recommended that a secondary database be used to store this information. Additionally, a system property is available to enable reading and writing to this database directly: `XL.UserProfileAuditInSecondaryDS`.

By default, this property is set to `false`. This means that there is no interaction with a secondary database. However, all historical reports are always queried from the secondary database. A restore of the transactional database should be done on the secondary data source on a regular basis. The advantage of this method is that restoring the database is simpler. On the other hand, more and more data might need to be restored in the long run and this method will take longer and longer to finish.

If this property is set to `true`, the system reads and writes all user profile data directly to and from the secondary database. The User Profile Audit interacts with the secondary database directly. However, other tables need to be replicated from the transactional database because the report needs them for access control and filtering of the report itself. These tables and constraints can be disabled for ease of data backup, restore, or replication, and are as follows:

- `AAD` — `ACT` (`ACT_`*`Key`*) `FK_AAD_FK_AAD_AC_ACT` (<*`Table_name`*> (<*`Referenced_column_name`*>) <*`Name of the FK constraint`*>)

  `UGP` (`UGP_`*`Key`*) `FK_AAD_FK_AAD_UG_UGP`

- ACT — ACT (Parent_*Key*) FK_ACT_ACT

  SRP (SRP_*Key*) FK_ACT_SRP

- POL
- REQ

  ORC (ORC_*Key*) FK_REQ_ORC

  OST (OST_*Key*) FK_REQ_OST

  USR (USR_*Key*) FK_REQ_USR

- UGP
- USG

  RUL (RUL_*Key*) FK_USG_RUL

  UGP (UGP_*Key*) FK_USG_UGP

  USR (USR_*Key*) FK_USG_USR

- USR

  ACT (ACT_*Key*) FK_USR_ACT

Steps to Set Up a Secondary Datasource

To set up a secondary database, do the following:

1. Create the secondary database by performing a backup and restore of the transactional database under a different database name, or by replicating the transactional database.
2. Setup the application server to use the secondary database (see the following sections).
3. Set the system property `XL.UserProfileAuditInSecondaryDS` to `True` if you want User Profile Audit data to go directly to the secondary database.
4. Set the daily restore or replication property to true for all the tables listed in the previous section. Otherwise, set up either a full restore or replication.
5. Make sure all stored procedures are also replicated correctly in the secondary database.

Note: Define the 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 Datasource

To create a new data source running 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, perform the following steps:

1. Edit the `xell-ds.xml` file by adding the following as a second `xa-datasource` tag for 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 will be 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`

   Remember to change the database name, user name, and password to connect to the database you set up as the secondary database.

2. Delete the `xlreportds-service.xml` file.
3. Restart the JBoss server.

> **Note:** Do not add the `xa-datasource` block given in this section and 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 a clustered configuration, ensure that the changes made to the `xell-ds.xml` file reflect on all machines in the cluster. Then restart the JBoss servers on all the machines.

In a standalone setup, both `xell-ds.xml` and `xlreportds-service.xml` are in the *`JBOSS_HOME`*`\server\default\deploy\` directory. However, in a clustered setup, the file `xell-ds.xml` is in the *`JBOSS_HOME`*`\server\all\farm\` directory, whereas `xlreportds-service.xml` in the *`JBOSS_HOME`*`\server\all\deploy\` directory.

Using WebLogic with a Secondary Datasource

Before changing the data source used by Oracle Identity Manager for reporting, a new data source needs to be created in WebLogic. Follow the WebLogic manuals to setup a new data source.

To configure WebLogic with a secondary data source, using Oracle Database, do the following:

1. Log in to the WebLogic administrative console and navigate to JDBC Connection Pools.
2. 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>`*
3. Create a secondary data source and deploy it on the server. To do this, 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`
4. Change the `weblogic.profile` file to point to the new data source. The weblogic.profile file is located at *`XL_HOME`*`/Xellerate/profiles.` Add the JNDI name as

   `datasource.report=jdbc/xlXAReportingDS`.
5. After modifiying the profile, run the patch command (`patch_weblogic`) for the changes to take effect.

> **Note:** For a WebLogic clustered configuration ,the secondary datasource must be deployed on all the cluster members participating in a cluster.

Using WebSphere with a Secondary Datasource

Before changing the data source used by Oracle Identity Manager for reporting, a new data source needs to be created in WebSphere. Follow the WebSphere manuals to set up a new data source.

To configure WebSphere with a secondary data source, using Oracle Database, do the following:

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. Modify the `websphere.profile` to add the JNDI information that points to the new data source in the *XL_HOME*`/xellerate/Profiles` directory. For this comment out the existing datasource 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. After modifiying the profile, run the `patch_websphere.cmd` or `patch_websphere.sh` as applicable from the *XL_HOME*`\xellerate\setup` directory.

Cluster Configuration

For a clustered set up, you need to individually modify the `websphere.profile` file on all nodes participating in the cluster. Then 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. Finally, stop and restart all nodes and servers.

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. First, 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_GROUPNAME");
                    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

W

X