Oracle® Identity Management

Application Developer's Guide

10*g* Release 2 (10.1.2)

Part No. B14087-01

December 2004

ORACLE®

Oracle Identity Management Application Developer's Guide, 10*g* Release 2 (10.1.2)

Part No. B14087-01

Primary Author: Henry Abrecht

Contributing Author: Jennifer Polk, Richard Smith

Contributor: Kamalendu Biswas, Ramakrishna Bollu, Saheli Dey, Bruce Ernst, Rajinder Gupta, Ashish Kolli, Stephen Lee, David Lin, Radhika Moolky, Samit Roy, David Saslav

This product contains SSLPlus Integration SuiteTM version 1.2, from Consensus Development Corporation.

iPlanet is a registered trademark of Sun Microsystems, Inc.

Contents

3 Developing Applications with Oracle Extensions to the Standard APIs

4 Developing Provisioning-Integrated Applications

5 Developing Directory Plug-ins

6 Integrating with Oracle Delegated Administration Services

7 Developing Applications for Single Sign-On

9 DBMS_LDAP PL/SQL Reference

B DSML Syntax

List of Figures

List of Tables

Send Us Your Comments

Oracle Identity Management Application Developer's Guide, 10*g* Release 2 (10.1.2)

Part No. B14087-01

Oracle welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the title and part number of the documentation and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: appserverdocs_us@oracle.com
- FAX: (650) 506-7375. Attn: Oracle Application Server Documentation Manager
- Postal service:

  Oracle Corporation
  Oracle Application Server Documentation
  500 Oracle Parkway, M/S 1op6
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, telephone number, and electronic mail address (optional).

If you have problems with the software, please contact your local Oracle Support Services.

Preface

Oracle Identity Management Application Developer's Guide explains how to modify applications to work with the Oracle Identity Management infrastructure. For the purposes of this book, this infrastructure consists of Oracle Application Server Single Sign-On, Oracle Internet Directory, Oracle Delegated Administration Services, and the Directory Integration Platform.

This preface contains these topics:

Audience

The following readers can benefit from this book:

- Developers who want to integrate applications with the Oracle Identity Management infrastructure. This process involves storing and updating information in an Oracle Internet Directory server. It also involves modifying applications to work with mod_osso, an authentication module on the Oracle HTTP Server.
- Anyone who wants to learn about the LDAP APIs and Oracle extensions to these APIs.

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. 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 additional information, visit the Oracle Accessibility Program Web site at

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

Accessibility of Code Examples in Documentation

JAWS, a Windows screen reader, 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, JAWS 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.

Structure

Part I, Programming for Oracle Identity Management

Chapter 1, "Developing Applications for Oracle Identity Management"

Takes a high level look at how to integrate applications with the Oracle Identity Management infrastructure. Introduces the reader to the Oracle Internet Directory Software Developer's Kit 10*g* Release 2 (10.1.2). Provides an overview of how an application can use the kit to integrate with the directory.

Chapter 2, "Developing Applications with Standard LDAP APIs"

Provides a brief overview of all of the major operations available in the C API and the PL/SQL API. Provides developers a general understanding of Lightweight Directory Access Protocol (LDAP) from a perspective independent of the API.

Chapter 3, "Developing Applications with Oracle Extensions to the Standard APIs"

Explains the concepts behind Oracle extensions to LDAP APIs. Describes the abstract entities that are modeled by the extensions as well as the usage model of the Oracle extensions.

Chapter 4, "Developing Provisioning-Integrated Applications"

Explains how to develop applications that can use the Oracle Directory Provisioning Integration Service in the Oracle Directory Integration and Provisioning platform. These applications can be either legacy applications or third-party applications that are based on the Oracle platform.

Chapter 5, "Developing Directory Plug-ins"

Explains how to use the plug-in framework for Oracle Internet Directory to facilitate custom development.

Chapter 6, "Integrating with Oracle Delegated Administration Services"

Explains how developers can use the DAS URL to integrate with Oracle Delegated Administration Services.

Chapter 7, "Developing Applications for Single Sign-On"

Explains how the HTTP authentication module mod_osso protects applications enabled by OracleAS Single Sign-On. Provides code that demonstrates how applications are integrated with mod_osso.

Part II Oracle Internet Directory API Reference

Chapter 8, "C API Reference"
Introduces the standard C API. Provides examples of how to use it.

Chapter 9, "DBMS_LDAP PL/SQL Reference"
Introduces the `DBMS_LDAP` package, which enables PL/SQL programmers to access data from LDAP servers. Provides examples of how to use `DBMS_LDAP`.

Chapter 10, "Java API Reference"
Directs readers to the Java APIs for Oracle Internet Directory. Provides a link to the standard API and a link to the Oracle extensions.

Chapter 11, "DBMS_LDAP_UTL PL/SQL Reference"
Contains reference material for the `DBMS_LDAP_UTL` package, which extends the `DBMS_LDAP` package.

Chapter 12, "DAS_URL Interface Reference"
Describes the Oracle extensions to the DAS_URL API.

Chapter 13, "Provisioning Integration API Reference"
Contains reference information for the Directory Integration and Provisioning Platform API.

Part III Appendixes

Appendix A, "Syntax for LDIF and Command-Line Tools"
Provides syntax, usage notes, and examples for using LDAP Data Interchange Format (LDIF) and LDAP command line tools

Appendix B, "DSML Syntax"
Provides syntax and usage notes for DSML (XML) integration.

Glossary
Defines terms used in this book.

Related Documents

For more information, see these Oracle resources:

- *Oracle Identity Management Concepts and Deployment Planning Guide*
- *Oracle Internet Directory Administrator's Guide*
- *Oracle Identity Management Integration Guide*
- *Oracle Identity Management Guide to Delegated Administration*
- *Oracle Application Server Single Sign-On Administrator's Guide*
- *PL/SQL User's Guide and Reference*
- *Oracle Database Application Developer's Guide - Fundamentals*

In North America, printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com/`

Customers in Europe, the Middle East, and Africa (EMEA) can purchase documentation from

`http://www.oraclebookshop.com/`

Other customers can contact their Oracle representative to purchase printed documentation.

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

`http://otn.oracle.com/admin/account/membership.html`

If you already have a user name and password for OTN, then you can go directly to the documentation section of the OTN Web site at

`http://otn.oracle.com/docs/index.htm`

To access the database documentation search engine directly, please visit

`http://tahiti.oracle.com`

For additional information, see:

- Chadwick, David. *Understanding X.500—The Directory.* Thomson Computer Press, 1996.
- Howes, Tim and Mark Smith. *LDAP: Programming Directory-enabled Applications with Lightweight Directory Access Protocol.* Macmillan Technical Publishing, 1997.
- Howes, Tim, Mark Smith and Gordon Good, *Understanding and Deploying LDAP Directory Services.* Macmillan Technical Publishing, 1999.
- Internet Assigned Numbers Authority home page, `http://www.iana.org`, for information about object identifiers
- Internet Engineering Task Force (IETF) documentation available at: `http://www.ietf.org`, especially:
  - The LDAPEXT charter and LDAP drafts
  - The LDUP charter and drafts
  - RFC 2254, "The String Representation of LDAP Search Filters"
  - RFC 1823, "The LDAP Application Program Interface"
- The OpenLDAP Community, `http://www.openldap.org`

Conventions

This section describes the conventions used in the text and code examples of this documentation set. It describes:

- Conventions in Text
- Conventions in Code Examples
- Conventions for Windows Operating Systems

Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	When you specify this clause, you create an **index-organized table**.
Italics	Italic typeface indicates book titles or emphasis.	*Oracle Database Concepts* Ensure that the recovery catalog and target database do *not* reside on the same disk.
`UPPERCASE monospace (fixed-width) font`	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, usernames, and roles.	You can specify this clause only for a `NUMBER` column. You can back up the database by using the `BACKUP` command. Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view. Use the `DBMS_STATS.GENERATE_STATS` procedure.
`lowercase monospace (fixed-width) font`	Lowercase monospace typeface indicates executable programs, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names and connect identifiers, user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values. *Note:* Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter `sqlplus` to start SQL*Plus. The password is specified in the `orapwd` file. Back up the datafiles and control files in the `/disk1/oracle/dbs` directory. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true`. Connect as `oe` user. The `JRepUtil` class implements these methods.
`lowercase italic monospace (fixed-width) font`	Lowercase italic monospace font represents placeholders or variables.	You can specify the *`parallel_clause`*. Run *`old_release`*`.SQL` where *`old_release`* refers to the release you installed prior to upgrading.

Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

Convention	Meaning	Example
`[ ]`	Anything enclosed in brackets is optional.	`DECIMAL (`*`digits`*` [ , `*`precision`*` ])`
`{ }`	Braces are used for grouping items.	`{ENABLE | DISABLE}`
`|`	A vertical bar represents a choice of two options.	`{ENABLE | DISABLE}` `[COMPRESS | NOCOMPRESS]`

Convention	Meaning	Example

`...`	Ellipsis points mean repetition in syntax descriptions.
	
	In addition, ellipsis points can mean an omission in code examples or text.
		`CREATE TABLE ... AS subquery;`
		
		`SELECT col1, col2, ... , coln FROM employees;`

Other symbols	You must use symbols other than brackets ([ ]), braces ({ }), vertical bars (|), and ellipsis points (...) exactly as shown.	`acctbal NUMBER(11,2);` `acct    CONSTANT NUMBER(4) := 3;`
Italics	Italicized text indicates placeholders or variables for which you must supply particular values.	`CONNECT SYSTEM/system_password` `DB_NAME = database_name`

`UPPERCASE`	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. Because these terms are not case sensitive, you can use them in either UPPERCASE or lowercase.	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

`lowercase`	Lowercase typeface indicates user-defined programmatic elements, such as names of tables, columns, or files.
	
	Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

Conventions for Windows Operating Systems

The following table describes conventions for Windows operating systems and provides examples of their use.

Convention	Meaning	Example
Choose **Start** > *menu item*	How to start a program.	To start the Database Configuration Assistant, choose **Start** > **Programs** > **Oracle** - *HOME_NAME* > **Configuration and Migration Tools** > **Database Configuration Assistant**.
File and directory names	File and directory names are not case sensitive. The following special characters are not allowed: left angle bracket (<), right angle bracket (>), colon (:), double quotation marks ("), slash (/), pipe (|), and dash (-). The special character backslash (\) is treated as an element separator, even when it appears in quotes. If the filename begins with \\, then Windows assumes it uses the Universal Naming Convention.	c:\winnt"\"system32 is the same as C:\WINNT\SYSTEM32
`C:\>`	Represents the Windows command prompt of the current hard disk drive. The escape character in a command prompt is the caret (^). Your prompt reflects the subdirectory in which you are working. Referred to as the *command prompt* in this manual.	`C:\oracle\oradata>`

Convention	Meaning	Example
Special characters	The backslash (\) special character is sometimes required as an escape character for the double quotation mark (") special character at the Windows command prompt. Parentheses and the single quotation mark (') do not require an escape character. Refer to your Windows operating system documentation for more information on escape and special characters.	`C:\>exp HR/HR TABLES=employees QUERY=\"WHERE job_id='SA_REP' and salary<8000\"`
HOME_NAME	Represents the Oracle home name. The home name can be up to 16 alphanumeric characters. The only special character allowed in the home name is the underscore.	`C:\> net start Oracle`*HOME_NAME*`TNSListener`
ORACLE_HOME and *ORACLE_BASE*	In releases prior to Oracle8*i* release 8.1.3, when you installed Oracle components, all subdirectories were located under a top level *ORACLE_HOME* directory. The default for Windows NT was `C:\orant`. This release complies with Optimal Flexible Architecture (OFA) guidelines. All subdirectories are not under a top level *ORACLE_HOME* directory. There is a top level directory called *ORACLE_BASE* that by default is `C:\oracle\product\10.1.0`. If you install the latest Oracle release on a computer with no other Oracle software installed, then the default setting for the first Oracle home directory is `C:\oracle\product\10.1.0\db_`*n*, where *n* is the latest Oracle home number. The Oracle home directory is located directly under *ORACLE_BASE*. All directory path examples in this guide follow OFA conventions. Refer to *Oracle Database Installation Guide for Windows* for additional information about OFA compliances and for information about installing Oracle products in non-OFA compliant directories.	Go to the *ORACLE_BASE*`\`*ORACLE_HOME*`\rdbms\admin` directory.

What's New in the SDK?

This document acquaints you with new features in the Oracle Internet Directory Software Developer's Kit—both in the present release and in the last release. Use the links provided to learn more about each feature.

New Features in the Release 10.1.2 SDK

The release 10.1.2 SDK adds these features:

- Dynamic password verifiers

  This feature addresses the needs of applications that provide parameters for password verifiers only at runtime. To learn more, see "Creating Dynamic Password Verifiers" in Chapter 3.

- Binary support for `ldapmodify`, `ldapadd`, and `ldapcompare` plug-ins

  Directory plug-ins can now access binary attributes in the directory database. To learn more, see "Binary Support in the Plug-in Framework" in Chapter 5.

New Features in the Release 9.0.4 SDK

The following features made their debut in the release 9.0.4 SDK:

- URL API for Oracle Delegated Administration Services

  This API enables you to build administrative and self-service consoles that delegated administrators can use to perform directory operations. To learn more, see Chapter 6.

- PL/SQL API Enhancements:
  - New functions in the LDAP v3 standard. Previously available only in the C API, these functions are now available in PL/SQL.
  - Functions that enable proxied access to middle-tier applications.
  - Functions that create and manage provisioning profiles in the directory integration and provisioning platform.

  To learn more, see Chapter 4.

- Plug-in support for external authentication

  This feature enables administrators to use Microsoft Active Directory to store and manage security credentials for Oracle components. To learn more, see Chapter 5.

- Server discovery using DNS

  This feature enables directory clients to discover the host name and port number of a directory server. It reduces the cost of maintaining directory clients in large deployments. To learn more, see "Server Discovery Functionality" in Chapter 2.

- XML support for the directory SDK and directory tools

  This feature enables LDAP tools to process XML as well as LDIF notation. Directory APIs can manipulate data in a DSML 1.0 format.

- Caching for client-side referrals

  This feature enables clients to cache referral information, speeding up referral processing. To learn more, see "LDAP Session Handle Options" in Chapter 6.

Part I

Programming for Oracle Identity Management

Part I shows you how to modify your applications to work with the different components of Oracle Identity Management. This section begins with an introduction to the Oracle Internet Directory SDK and to LDAP programming concepts. You then learn how to use the three LDAP APIs and their extensions to enable applications for Oracle Internet Directory. The section ends with the tasks required to enable an application for single sign-on.

Part I contains these chapters:

1

Developing Applications for Oracle Identity Management

Oracle Identity Management provides a shared infrastructure for all Oracle applications. It also provides services and interfaces that facilitate third-party enterprise application development. These interfaces are useful for application developers who need to incorporate identity management into their applications.

This chapter discusses these interfaces and recommends application development best practices in the Oracle Identity Management environment.

There are two types of applications that can be integrated with Oracle Identity Management:

- Existing applications already used in the enterprise. The enterprise might have already invested in such applications and would benefit from their integration with the Oracle Identity Management infrastructure.
- New applications being developed by corporate IT departments or Slavs that are based on the Oracle technology stack

This chapter contains the following topics:

Benefits of Integrating with Oracle Identity Management

Enterprise applications integrating with the Oracle Identity Management infrastructure receive the following benefits:

- **Integration facilitates faster application deployment with lower costs**: Enterprises (primarily Oracle customers) already using an existing Oracle Identity Management infrastructure can deploy new applications using the self-service console of Oracle Delegated Administration Services. Delegating application administration to users reduces the deployment cost of the application.
- **Seamless integration with Oracle applications**: Because all Oracle applications rely on the Oracle Identity Management infrastructure, new enterprise applications can use all the features Oracle Identity Management offers.

- **Seamless integration with third-party identity management solutions**: Because the Oracle Identity Management infrastructure already has built-in capabilities for integrating with third-party identity management solutions, application developers can take advantage of the identity management features.

Oracle Identity Management Services Available for Application Integration

Custom applications can use Oracle Identity Management through a set of documented and supported services and Opes. For example:

- Oracle Internet Directory provides LDAP APIs for C, Java, and PL/SQL, and is compatible with other LDAP SDKs.
- Oracle Delegated Administration Services provides a core self-service console that can be customized to support third-party applications. In addition, they provide a number of services for building customized administration interfaces that manipulate directory data.
- Oracle Directory Integration Services facilitate the development and deployment of custom solutions for synchronizing Oracle Internet Directory with third-party directories and other user repositories.
- Oracle Provisioning Integration Services provide a mechanism for provisioning third-party applications, as well as a means of integrating the Oracle environment with other provisioning systems.
- OracleAS Single Sign-On provides APIs for developing and deploying partner applications that share a single sign-on session with other Oracle Web applications.
- JAZN is the Oracle implementation of the Oracle Application Server Java Authentication and Authorization Service (JAAS) Support standard that allows applications developed for the Web using the Oracle J2EE environment to use the identity management infrastructure for authentication and authorization.

Integrating Existing Applications with Oracle Identity Management

An enterprise may have already deployed certain applications to perform critical business functions. The Oracle Identity Management infrastructure provides the following services that can be leveraged by the deployment to modify existing applications:

- **Automated User Provisioning**: The deployment can develop a custom provisioning agent that automates the provisioning of users in the existing application in response to provisioning events in the Oracle Identity Management infrastructure. This agent must be developed using the interfaces of Oracle Provisioning Integration Service.

  See Also: *Oracle Internet Directory Administrator's Guide* for more information about developing automated user provisioning

- **User Authentication Services**: If the user interface of the existing application is based on HTTP, integrating it with Oracle HTTP Server and protecting its URL using `mod_osso` will authenticate all incoming user requests using the OracleAS Single Sign-On service.
- **Centralized User Profile Management**: If the user interface of the existing application is based on HTTP, and it is integrated with OracleAS Single Sign-On for authentication, the application can use the self-service console of Oracle

Delegated Administration Services to enable centralized user profile management. The self-service console can be customized by the deployment to address the specific needs of the application.

Integrating New Applications with Oracle Identity Management

Application developers can use the services provided by the Oracle Identity Management infrastructure more extensively if they are developing a new application or planning a new release of an existing application. Application developers should consider the following integration points:

- **User Authentication Services**: The application developer has the following options:
  - If the application is based on J2EE, it can use the services provided by the Oracle Application Server Java Authentication and Authorization Service (JAAS) Support interface.
  - If the application relies on Oracle Application Server Containers for J2EE, it can use the services provided by `mod_osso` to authenticate users and obtain important information about the user in the HTTP headers.
  - If the application is a standalone Web-based application, it can use OracleAS Single Sign-On as a partner application using the OracleAS Single Sign-On APIs.
  - If the application provides an interface that is not Web-based, it can use the Oracle Internet Directory LDAP APIs (available in C, PL/SQL and Java) to authenticate users.
- **Centralized Profile Management**: The application developer has the following options available:
  - The application developer can model application-specific profiles and user preferences as attributes in Oracle Internet Directory.
  - If the user interface of the application is based on HTTP, and it is integrated with OracleAS Single Sign-On for authentication, the application can leverage the self-service console of Oracle Delegated Administration Services to enable centralized user profile management. The self-service console can be customized by the deployment to address the specific needs of the application.
  - The application can also retrieve user profiles at run time using the Oracle Internet Directory LDAP APIs (available in C, PL/SQL and Java).
- **Automated User Provisioning**: Application developers should consider the following options:
  - If the user interface of the application is based on HTTP and it is integrated with OracleAS Single Sign-On for authentication, then the application developer can implement automated user provisioning the first time a user accesses the application
  - The application can also be integrated with the Oracle Internet Directory Provisioning Integration Service, which enables it to automatically provision or de-provision user accounts in response to administrative actions, such as adding an identity, modifying the properties of an existing identity, or deleting an existing identity in the Oracle Identity Management infrastructure

See Also: *Oracle Identity Management Integration Guide*

Integrating J2EE Applications with Oracle Identity Management

Oracle Application Server Containers for J2EE (OC4J) provides standards-based J2EE security support. Furthermore, J2EE applications deployed within OC4J can be authenticated and authorized against Oracle Identity Management.

J2EE security provides standard APIs such as `getUserPrincipal` and `isUserInRole` that enable applications to obtain information about the authenticated user. If an application requires additional user information (or attributes), it can use Oracle Internet Directory LDAP APIs to retrieve this information from the directory.

To learn more about J2EE and JAAS security, see *Oracle Application Server Containers for J2EE Security Guide*.

Directory Programming: An Overview

This section introduces you to the Oracle Internet Directory Software Developer's Kit. It provides an overview of how an application can use the kit to integrate with the directory. You are also acquainted with the rest of the directory product suite.

The section contains these topics:

- Programming Languages Supported by the SDK
- SDK Components
- Application Development in the Directory Environment
- Other Components of Oracle Internet Directory

Programming Languages Supported by the SDK

The SDK is for application developers who use C, C++, and PL/SQL. Java developers must use the JNDI provider from Sun Microsystems to integrate with the directory.

SDK Components

Oracle Internet Directory Software Developer's Kit 10*g* Release 2 (10.1.2) consists of the following:

- A C API compliant with LDAP Version 3
- A PL/SQL API contained in a PL/SQL package called `DBMS_LDAP`
- Sample programs
- *Oracle Identity Management Application Developer's Guide* (this document)
- Command-line tools

Application Development in the Directory Environment

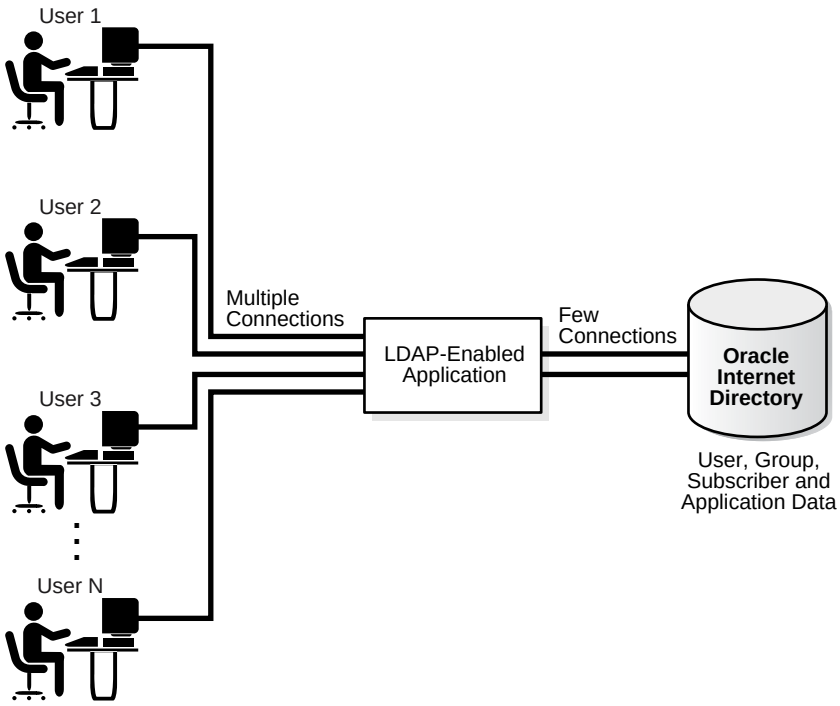
This section contains these topics:

- Architecture of a Directory-Enabled Application
- Directory Interactions During the Application Life Cycle
- Services and APIs for Integrating Applications with Oracle Internet Directory
- Integrating Existing Applications with Oracle Internet Directory
- Integrating New Applications with Oracle Internet Directory

Architecture of a Directory-Enabled Application

Most directory-enabled applications are backend programs that simultaneously handle multiple requests from multiple users. Figure 1–1 shows how a directory is used by such applications.

Figure 1–1 A Directory-Enabled Application



This illustration shows four hypothetical users connecting to a middle tier. Each user has a connection, for a total of four connections. The middle tier then connects to Oracle Internet Directory by using only two connections. The directory contains data for groups, subscribers, and applications.

****

As Figure 1–1 shows, when a user request involves an LDAP-enabled operation, the application processes the request using a smaller set of pre-created directory connections.

Directory Interactions During the Application Life Cycle

Table 1–1 on page 1-6 walks you through the directory operations that an application typically performs during its lifecycle.

Table 1–1 Interactions During Application Lifecycle

Point in Application Lifecycle	Logic
Application Installation	1. Create an application identity in the directory. The application uses this identity to perform most of its LDAP operations. 2. Give the application identity LDAP authorizations by making it part of the correct LDAP groups. These authorizations enable the application to accept user credentials and authenticate them against the directory. The directory can also use application authorizations to proxy for the user when LDAP operations must be performed on the user's behalf.
Application Startup and Bootstrap	The application must retrieve credentials that enable it to authenticate itself to the directory. If the application stores configuration metadata in Oracle Internet Directory, it can retrieve that metadata and initialize other parts of the application. The application can then establish a pool of connections to serve user requests.
Application Runtime	For every end-user request that needs an LDAP operation, the application can: ■ Pick a connection from the pool of LDAP connections ■ Switch the user to the end-user identity if the LDAP operation needs to be performed with the effective rights of the end-user ■ Perform the LDAP operation by using either the regular API or the API enhancements described in this chapter ■ Ensure that the effective user is now the application identity once the LDAP operation is complete ■ Return the LDAP connection back to the pool of connections
Application Shutdown	Abandon any outstanding LDAP operations and close all LDAP connections.
Application Deinstallation	Remove the application identity and the LDAP authorizations granted to it.

Services and APIs for Integrating Applications with Oracle Internet Directory

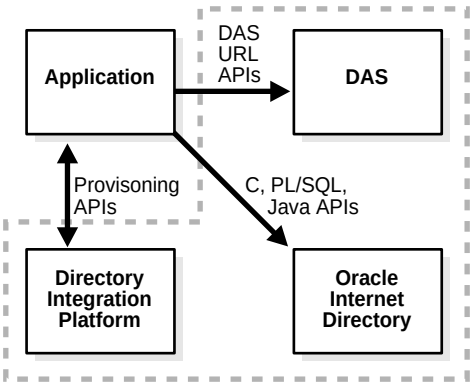
Application developers can integrate with Oracle Internet Directory by using the services and APIs listed and described in Table 1–2 on page 1-7.

Table 1–2 Services and APIs for Integrating with Oracle Internet Directory

Service/API	Description	More Information
Standard LDAP APIs in C, PL/SQL and Java	These provide basic LDAP operations. The standard LDAP API used in Java is the JNDI API with the LDAP service provider from Sun Microsystems.	Chapter 2, "Developing Applications with Standard LDAP APIs"
Oracle Extensions to Standard C, PL/SQL and Java APIs	These APIs provide programmatic interfaces that model various concepts related to identity management.	Chapter 3, "Developing Applications with Oracle Extensions to the Standard APIs"
Oracle Delegated Administration Services	Oracle Delegated Administration Services consists of a self-service console and administrative interfaces. You can modify the administrative interfaces to support third-party applications.	▪ Chapter 6, "Integrating with Oracle Delegated Administration Services" ▪ The chapter about the delegated administration services framework in *Oracle Identity Management Guide to Delegated Administration*
Oracle Directory Provisioning Integration Service	You can use the Oracle Provisioning Integration System to provision third-party applications and integrate other provisioning systems.	▪ Chapter 4, "Developing Provisioning-Integrated Applications" ▪ The introductory chapter in *Oracle Identity Management Integration Guide*
Oracle Internet Directory Plug-ins	You can use plug-ins to customize directory behavior in certain deployments.	▪ Chapter 5, "Developing Directory Plug-ins" ▪ The chapter about plug-ins in *Oracle Internet Directory Administrator's Guide*

Figure 1–2 shows an application leveraging some of the services illustrated in Table 1–2 on page 1-7.

Figure 1–2 An Application Leveraging APIs and Services



As Figure 1–2 shows, the application integrates with Oracle Internet Directory as follows:

- Using PL/SQL, C, or Java APIs, it performs LDAP operations directly against the directory.

- In some cases, it directs users to self-service features of Oracle Delegated Administration Services.
- It is notified of changes to entries for users or groups in Oracle Internet Directory. The Oracle Directory Provisioning Integration Service provides this notification.

Integrating Existing Applications with Oracle Internet Directory

Your enterprise may already have deployed applications that you may have wanted to integrate with the Oracle identity management infrastructure. You can still integrate these applications using the services presented in Table 1–3.

Table 1–3 Services for Modifying Existing Applications

Service	Description	More Information
Automated User Provisioning	You can develop an agent that automatically provisions users when provisioning events occur in the Oracle identity management infrastructure. You use interfaces of the Oracle Directory Provisioning Integration Service to develop this agent.	Chapter 4, "Developing Provisioning-Integrated Applications"
User Authentication Services	If your user interface is based on HTTP, you can integrate it with the Oracle HTTP Server. This enables you to use mod_osso and OracleAS Single Sign-On to protect the application URL.	*Oracle Application Server Single Sign-On Administrator's Guide*
Centralized User Profile Management	If your user interface is based on HTTP and is integrated with OracleAS Single Sign-On, you can use the Oracle Internet Directory Self-Service Console to manage user profiles centrally. You can tailor the console to the needs of your application.	■ Chapter 6, "Integrating with Oracle Delegated Administration Services" ■ The chapter about the delegated administration services framework in *Oracle Identity Management Guide to Delegated Administration*

Integrating New Applications with Oracle Internet Directory

If you are developing a new application or planning a new release of an existing application, you have many directory integration options at your disposal. Table 1–4 on page 1-9 lists and describes these.

Table 1–4 Application Integration Points

Integration Point	Available Options	More Information
User Authentication Services	If your application is based on J2EE, it can use the JAZN interface to authenticate users. If it relies on OC4J, it can use mod_osso for the same purpose. The second option enables the application to obtain information about the user from HTTP headers. If your application is Web based and standalone, it can still integrate with OracleAS Single Sign-On, then it can still leverage Oracle Application Server Single Sign-On by becoming a partner application using the single sign-on APIs. Finally, if the application provides a non-Web user interface, it can use the Oracle Internet Directory LDAP APIs to integrate users.	■ *Oracle Application Server Containers for J2EE User's Guide* ■ *Oracle Application Server Single Sign-On Administrator's Guide* ■ Part II, "Oracle Internet Directory Programming Reference". This section is devoted to the various LDAP APIs.
User Authorization Services	If your application is based on J2EE, it can use the JAZN interface to implement and enforce user authorizations for application resources. The application can define authorizations as groups in Oracle Internet Directory and can then check the authorizations of a user by checking his or her group membership. It can use the Oracle Internet Directory LDAP APIs for this purpose.	■ *Oracle Application Server Containers for J2EE User's Guide* ■ Part II, "Oracle Internet Directory Programming Reference". This section is devoted to the various LDAP APIs.
Centralized Profile Management	You can define application-specific profiles and user preferences as attributes in Oracle Internet Directory. If your user interface is based on HTTP and is integrated with OracleAS Single Sign-On, you can use the Oracle Internet Directory Self-Service Console to manage user profiles centrally. You can tailor the console to the needs of your application. Additionally, you can use the Oracle Internet Directory LDAP APIs to retrieve user profiles at runtime.	■ The chapter about deployment considerations in *Oracle Internet Directory Administrator's Guide* ■ Chapter 6, "Integrating with Oracle Delegated Administration Services" ■ *Oracle Identity Management Guide to Delegated Administration* ■ Part II of this guide, which is devoted to the various LDAP APIs
Automated User Provisioning	If your user interface is based on HTTP and it is integrated with OracleAS Single Sign-On, you can implement automated user provisioning the very first time a user accesses the application. You use the Oracle Directory Provisioning Integration Service to integrate the application with the Oracle identity management infrastructure. Once integrated, the application can provision or deprovision user accounts automatically when an administrator adds, modifies, or deletes an identity.	Chapter 4, "Developing Provisioning-Integrated Applications"

Other Components of Oracle Internet Directory

The SDK is just one component in the directory suite. Here are the others:

- Oracle directory server, LDAP Version 3
- Oracle directory replication server
- Oracle Directory Manager, a Java-based graphical user interface

- Oracle Internet Directory bulk tools
- *Oracle Internet Directory Administrator's Guide*

2

Developing Applications with Standard LDAP APIs

This chapter takes a high-level look at the operations that the standard LDAP API enables. It explains how to integrate your applications with the API. Before presenting these topics, the chapter revisits the Lightweight Directory Access Protocol (LDAP).

This chapter contains these topics:

History of LDAP

LDAP began as a lightweight front end to the X.500 Directory Access Protocol. LDAP simplifies the X.500 Directory Access Protocol in the following ways:

- It uses TCP/IP connections. These are lightweight compared to the OSI communication stack required by X.500 implementations
- It eliminates little-used and redundant features of the X.500 Directory Access Protocol
- It uses simple formats to represent data elements. These formats are easier to process than the complicated and highly structured representations in X.500.
- It uses a simplified version of the X.500 encoding rules used to transport data over networks.

LDAP Models

LDAP uses four basic models to define its operations:

- Naming Model
- Information Model

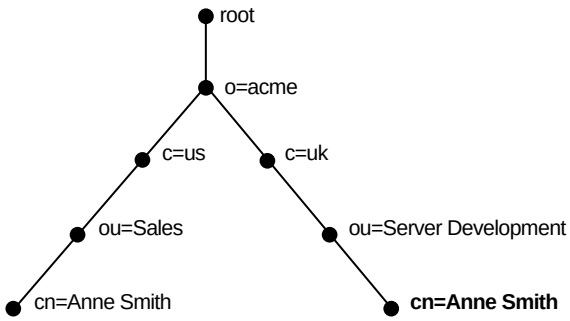
- Functional Model
- Security Model

Naming Model

The LDAP naming model enables directory information to be referenced and organized. Each entry in a directory is uniquely identified by a distinguished name (DN). The DN tells you exactly where an entry resides in the directory hierarchy. A directory information tree (DIT) is used to represent this hierarchy.

Figure 2–1 illustrates the relationship between a distinguished name and a directory information tree.

Figure 2–1 A Directory Information Tree



The DIT in Figure 2–1 shows entries for two employees of Acme Corporation who are both named Anne Smith. It is structured along geographical and organizational lines. The Anne Smith represented by the left branch works in the Sales division in the United States. Her counterpart works in the Server Development division in the United Kingdom.

The Anne Smith represented by the right branch has the common name (`cn`) Anne Smith. She works in an organizational unit (`ou`) named Server Development, in the country (`c`) of Great Britain (`uk`), in the organization (`o`) Acme. The DN for this Anne Smith entry looks like this:

```
cn=Anne Smith,ou=Server Development,c=uk,o=acme
```

Note that the conventional format for a distinguished name places the lowest DIT component at the left. The next highest component follows, on up to the root.

Within a distinguished name, the lowest component is called the relative distinguished name (RDN). In the DN just presented, the RDN is `cn=Anne Smith`. The RDN for the entry immediately above Anne Smith's RDN is `ou=Server Development`. And the RDN for the entry immediately above `ou=Server Development` is `c=uk`, and so on. A DN is thus a sequence of RDNs separated by commas.

To locate a particular entry within the overall DIT, a client uniquely identifies that entry by using the full DN—not simply the RDN—of that entry. To avoid confusion between the two Anne Smiths in the global organization depicted in Figure 2–1, you use the full DN for each. If there are two employees with the same name in the same organizational unit, you can use other mechanisms. You may, for example, use a unique identification number to identify these employees.

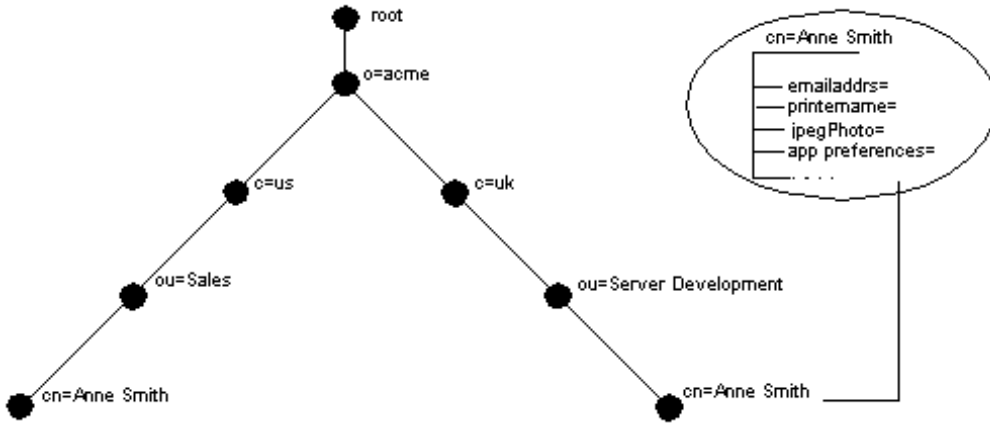
Information Model

The LDAP information model determines the form and character of information in the directory. This model uses the concept of entries as its defining characteristic. In a directory, an entry is a collection of information about an object. A telephone directory, for example, contains entries for people. A library card catalog contains entries for books. An online directory may contain entries for employees, conference rooms, e-commerce partners, or shared network resources such as printers.

In a typical telephone directory, a person entry contains an address and a phone number. In an online directory, each of these pieces of information is called an attribute. A typical employee entry contains attributes for a job title, an e-mail address, and a phone number.

In Figure 2–2, the entry for Anne Smith in Great Britain (uk) has several attributes. Each provides specific information about her. Those listed in the balloon to the right of the tree are `emailaddrs`, `printername`, `jpegPhoto`, and `app preferences`. Note that the rest of the bullets in Figure 2–2 are also entries with attributes, although these attributes are not shown.

Figure 2–2 Attributes of the Entry for Anne Smith



Each attribute consists of an attribute type and one or more attribute values. The attribute type is the kind of information that the attribute contains—`jobTitle`, for instance. The attribute value is the actual information. The value for the `jobTitle` attribute, for example, might be `manager`.

Functional Model

The LDAP functional model determines what operations can be performed on directory entries. Table 2–1 on page 2-4 lists and describes the three types of functions:

Table 2–1 LDAP Functions

Function	Description
Search and read	The read operation retrieves the attributes of an entry whose name is known. The list operation enumerates the children of a given entry. The search operation selects entries from a defined area of the tree based on some selection criteria known as a search filter. For each matching entry, a requested set of attributes (with or without values) is returned. The searched entries can span a single entry, an entry's children, or an entire subtree. Alias entries can be followed automatically during a search, even if they cross server boundaries. An abandon operation is also defined, allowing an operation in progress to be canceled.
Modify	This category defines four operations that modify the directory: ■ Modify—change existing entries. You can add and delete values. ■ Add—insert entries into the directory ■ Delete—remove entries from the directory ■ Modify RDN—change the name of an entry
Authenticate	This category defines a bind operation. A bind enables a client to initiate a session and prove its identity to the directory. Oracle Internet Directory supports several authentication methods, from simple clear-text passwords to public keys. The unbind operation is used to terminate a directory session.

Security Model

The LDAP security model enables directory information to be secured. This model has several parts:

- Authentication

  Ensuring that the identities of users, hosts, and clients are correctly validated

- Access Control and Authorization

  Ensuring that a user reads or updates only the information for which that user has privileges

- Data Integrity: Ensuring that data is not modified during transmission

- Data Privacy

  Ensuring that data is not disclosed during transmission

- Password Policies

  Setting rules that govern how passwords are used

Authentication

Authentication is the process by which the directory server establishes the identity of the user connecting to the directory. Directory authentication occurs when an LDAP bind operation establishes an LDAP session. Every session has an associated user identity, also referred to as an authorization ID.

Oracle Internet Directory provides three authentication options: anonymous, simple, and SSL.

Anonymous Authentication If your directory is available to everyone, users may log in anonymously. In anonymous authentication, users leave the user name and password fields blank when they log in. They then exercise whatever privileges are specified for anonymous users.

Simple Authentication In simple authentication, the client uses an unencrypted DN and password to identify itself to the server. The server verifies that the client's DN and password match the DN and password stored in the directory.

Authentication Using Secure Sockets Layer (SSL) Secure Socket Layer (SSL) is an industry standard protocol for securing network connections. It uses a certificate exchange to authenticate users. These certificates are verified by trusted certificate authorities. A certificate ensures that an entity's identity information is correct. An entity can be an end user, a database, an administrator, a client, or a server. A certificate authority (CA) is an application that creates public key certificates that are given a high level of trust by all parties involved.

You can use SSL in one of the three authentication modes presented in Table 2–2.

Table 2–2 SSL Authentication Modes

SSL Mode	Description
No authentication	Neither the client nor the server authenticates itself to the other. No certificates are sent or exchanged. In this case, only SSL encryption and decryption are used.
One-way authentication	Only the directory server authenticates itself to the client. The directory server sends the client a certificate verifying that the server is authentic.
Two-way authentication	Both client and server authenticate themselves to each other, exchanging certificates.

In an Oracle Internet Directory environment, SSL authentication between a client and a directory server involves three basic steps:

1. The user initiates an LDAP connection to the directory server by using SSL on an SSL port. The default SSL port is `636`.
2. SSL performs the handshake between the client and the directory server.
3. If the handshake is successful, the directory server verifies that the user has the appropriate authorization to access the directory.

> **See Also:** *Oracle Advanced Security Administrator's Guide* for more information about SSL

Access Control and Authorization

The authorization process ensures that a user reads or updates only the information for which he or she has privileges. The directory server ensures that the user—identified by the authorization ID associated with the session—has the requisite permissions to perform a given directory operation. Absent these permissions, the operation is disallowed.

The mechanism that the directory server uses to ensure that the proper authorizations are in place is called access control. And an access control information item (ACI) is the directory metadata that captures the administrative policies relating to access control.

An ACI is stored in Oracle Internet Directory as user-modifiable operational attributes. Typically a whole list of these ACI attribute values is associated with a directory object. This list is called an access control list (ACL). The attribute values on that list govern the access policies for the directory object.

ACIs are stored as text strings in the directory. These strings must conform to a well-defined format. Each valid value of an ACI attribute represents a distinct access control policy. These individual policy components are referred to as ACI Directives or ACIs and their format is called the ACI Directive format.

Access control policies can be prescriptive: their security directives can be set to apply downward to all entries at lower positions in the directory information tree (DIT). The point from which an access control policy applies is called an access control policy point (ACP).

Data Integrity

Oracle Internet Directory uses SSL to ensure that data is not modified, deleted, or replayed during transmission. This feature uses cryptographic checksums to generate a secure message digest. The checksums are created using either the MD5 algorithm or the Secure Hash Algorithm (SHA). The message digest is included in each network packet.

Data Privacy

Oracle Internet Directory uses public-key encryption over SSL to ensure that data is not disclosed during transmission. In public-key encryption, the sender of a message encrypts the message with the public key of the recipient. Upon delivery, the recipient decrypts the message using his or her private key. The directory supports two levels of encryption:

- DES40

  The DES40 algorithm, available internationally, is a DES variant in which the secret key is preprocessed to provide forty effective key bits. It is designed for use by customers outside the USA and Canada who want to use a DES-based encryption algorithm.

- RC4_40

  Oracle is licensed to export the RC4 data encryption algorithm with a 40-bit key size to virtually all destinations where Oracle products are available. This makes it possible for international corporations to safeguard their entire operations with fast cryptography.

Password Policies

A password policy is a set of rules that govern how passwords are used. When a user attempts to bind to the directory, the directory server uses the password policy to ensure that the password provided meets the various requirements set in that policy.

When you establish a password policy, you set the following types of rules, to mention just a few:

- The maximum length of time a given password is valid
- The minimum number of characters a password must contain
- The ability of users to change their passwords

About the Standard LDAP APIs

The standard LDAP APIs enable you to perform the fundamental LDAP operations described in "LDAP Models". These APIs are available in C, PL/SQL, and Java. The first two are part of the directory SDK. The last is part of the JNDI package provided by Sun Microsystems. All three use TCP/IP connections. They are based on LDAP Version 3, and they support SSL connections to Oracle Internet Directory.

This section contains these topics:

- API Usage Model
- Getting Started with the C API
- Getting Started with the Java API
- Getting Started with the DBMS_LDAP Package

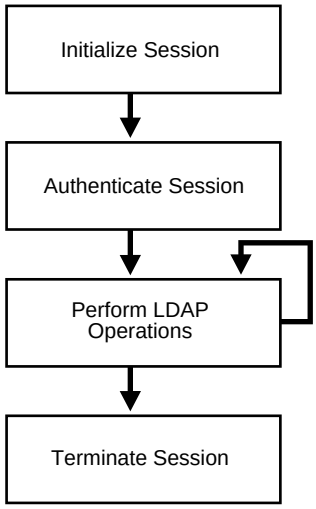
API Usage Model

Typically, an application uses the functions in the API in four steps:

1. Initialize the library and obtain an LDAP session handle.
2. Authenticate to the LDAP server if necessary.
3. Perform some LDAP operations and obtain results and errors, if any.
4. Close the session.

Figure 2–3 illustrates these steps.

Figure 2–3 Steps in Typical DBMS_LDAP Usage



Getting Started with the C API

When you build applications with the C API, you must include the header file `ldap.h`, located at *`ORACLE_HOME`*`/ldap/public`. In addition, you must dynamically link to the library located at *`ORACLE_HOME`*`/lib/libclntsh.so.10.1`.

> **See Also:** "Sample C API Usage" on page 8-41 to learn how to use the SSL and non-SSL modes

Getting Started with the DBMS_LDAP Package

The DBMS_LDAP package enables PL/SQL applications to access data located in enterprise-wide LDAP servers. The names and syntax of the function calls are similar to those of the C API. These functions comply with current recommendations of the Internet Engineering Task Force (IETF) for the C API. Note though that the PL/SQL API contains only a subset of the functions available in the C API. Most notably, only synchronous calls to the LDAP server are available in the PL/SQL API.

To begin using the PL/SQL LDAP API, use this command sequence to load DBMS_LDAP into the database:

1. Log in to the database, using SQL*Plus. Run the tool in the Oracle home in which your database is present. Connect as SYSUSER.

   ```
   SQL> CONNECT / AS SYSDBA
   ```

2. Load the API into the database, using this command:

   ```
   SQL> @?/rdbms/admin/catladap.sql
   ```

Getting Started with the Java API

Java developers can use the Java Naming and Directory Interface (JNDI) from Sun Microsystems to gain access to information in Oracle Internet Directory. The JNDI is found at this link:

`http://java.sun.com/products/jndi`

Although no Java APIs are provided in this chapter, the section immediately following, "Initializing the Session by Using JNDI", shows you how to use wrapper methods for the Sun JNDI to establish a basic connection.

Initializing an LDAP Session

All LDAP operations based on the C API require clients to establish an LDAP session with the LDAP server. For LDAP operations based on the PL/SQL API, a database session must first initialize and open an LDAP session. Most Java operations require a Java Naming and Directory Interface (JNDI) connection. The `oracle.ldap.util.jndi` package, provided here, simplifies the work involved in achieving this connection.

The section contains the following topics:

- Initializing the Session by Using the C API
- Initializing the Session by Using DBMS_LDAP
- Initializing the Session by Using JNDI

Initializing the Session by Using the C API

The C function `ldap_init()` initializes a session with an LDAP server. The server is not actually contacted until an operation is performed that requires it, allowing various options to be set after initialization.

`ldap_init` has the following syntax:

```
LDAP *ldap_init
(
const char      *hostname,
int             portno
);
```

Table 2–3 lists and defines the function parameters.

Table 2–3 Parameters for ldap_init()

Parameter	Description
`hostname`	Contains a space-separated list of directory host names or IP addresses represented by dotted strings. You can pair each host name with a port number as long as you use a colon to separate the two. The hosts are tried in the order listed until a successful connection is made. Note: A suitable representation for including a literal IPv6[10] address in the host name parameter is desired, but has not yet been determined or implemented in practice.
`portno`	Contains the TCP port number of the directory you would like to connect to. The default LDAP port of `389` can be obtained by supplying the constant `LDAP_PORT`. If a host includes a port number, this parameter is ignored.

`ldap_init()` and `ldap_open()` both return a session handle, or pointer, to an opaque structure that must be passed to subsequent calls to the session. These routines return `NULL` if the session cannot be initialized. You can check the error reporting mechanism for your operating system to determine why the call failed.

Initializing the Session by Using DBMS_LDAP

In the PL/SQL API, the function `DBMS_LDAP.init()` initiates an LDAP session. This function has the following syntax:

```
FUNCTION init (hostname IN VARCHAR2, portnum  IN PLS_INTEGER )
RETURN SESSION;
```

The function `init` requires a valid host name and port number to establish an LDAP session. It allocates a data structure for this purpose and returns a handle of the type `DBMS_LDAP.SESSION` to the caller. The handle returned from the call should be used in all subsequent LDAP operations defined by DBMS_LDAP for the session. The API uses these session handles to maintain state about open connections, outstanding requests, and other information.

A single database session can obtain as many LDAP sessions as required, although the number of simultaneous active connections is limited to 64. One database session typically has multiple LDAP sessions when data must be obtained from multiple servers simultaneously or when open sessions that use multiple LDAP identities are required.

Note: The handles returned from calls to `DBMS_LDAP.init()` are dynamic constructs: They do not persist across multiple database sessions. Attempting to store their values in a persistent form, and to reuse stored values at a later stage, can yield unpredictable results.

Initializing the Session by Using JNDI

The `oracle.ldap.util.jndi` package supports basic connections by providing wrapper methods for the JNDI implementation from Sun Microsystems. If you want to use the JNDI to establish a connection, see the following link:

`http://java.sun.com/products/jndi`

Here is an implementation of `oracle.ldap.util.jndi` that establishes a non-SSL connection:

```
import oracle.ldap.util.jndi
import javax.naming.*;

public static void main(String args[])
{
  try{
       InitialDirContext ctx = ConnectionUtil.getDefaultDirCtx(args[0], // host
                                                               args[1],  // port
                                                               args[2],  // DN
                                                               args[3];  // password)
       // Do work
     }
  catch(NamingException ne)
  {
    // javax.naming.NamingException is thrown when an error occurs
  }
}
```

Note:

- *DN* and *password* represent the bind DN and password. For anonymous binds, set these to "".
- You can use `ConnectionUtil.getSSLDirCtx()` to establish a no-authentication SSL connection.

Authenticating an LDAP Session

Individuals or applications seeking to perform operations against an LDAP server must first be authenticated. If the `dn` and `passwd` parameters of these entities are null, the LDAP server assigns a special identity, called anonymous, to these users. Typically, the anonymous user is the least privileged user of the directory.

Once a bind operation is complete, the directory server remembers the new identity until another bind occurs or the LDAP session terminates (`unbind_s`). The LDAP server uses the identity to enforce the security model specified by the enterprise in which it is deployed. The identity helps the LDAP server determine whether the user or application identified has sufficient privileges to perform search, update, or compare operations in the directory.

Note that the password for the bind operation is sent over the network in clear text. If your network is not secure, consider using SSL for authentication and other LDAP operations that involve data transfer.

This section contains these topics:

- Authenticating an LDAP Session by Using the C API
- Authenticating an LDAP Session by Using DBMS_LDAP

Authenticating an LDAP Session by Using the C API

The C function `ldap_simple_bind_s()` enables users and applications to authenticate to the directory server using a DN and password.

The function `ldap_simple_bind_s()` has this syntax:

```
int ldap_simple_bind_s
(
LDAP*ld,
char*dn,
char*passwd,
);
```

Table 2–4 lists and describes the parameters for this function.

Table 2–4 Arguments for ldap_simple_bind_s()

Argument	Description
`ld`	A valid LDAP session handle.
`dn`	The identity that the application uses for authentication.
`passwd`	The password for the authentication identity.

If the `dn` and `passwd` parameters for are `NULL`, the LDAP server assigns a special identity, called anonymous, to the user or application.

Authenticating an LDAP Session by Using DBMS_LDAP

The PL/SQL function `simple_bind_s` enables users and applications to use a DN and password to authenticate to the directory. `simple_bind_s` has this syntax:

```
FUNCTION simple_bind_s ( ld IN SESSION, dn IN VARCHAR2, passwd IN VARCHAR2)
RETURN PLS_INTEGER;
```

Note that this function requires as its first parameter the LDAP session handle obtained from `init`.

The following PL/SQL code snippet shows how the PL/SQL initialization and authentication functions just described might be implemented.

```
DECLARE
retvalPLS_INTEGER;
my_sessionDBMS_LDAP.session;

BEGIN
retval:= -1;
-- Initialize the LDAP session
my_session:= DBMS_LDAP.init('yow.acme.com',389);
--Authenticate to the directory
retval:=DBMS_LDAP.simple_bind_s(my_session, 'cn=orcladmin',
'welcome');
```

In the previous example, an LDAP session is initialized on the LDAP server `yow.acme.com`. This server listens for requests at TCP/IP port number `389`. The identity `cn=orcladmin`, whose password is `welcome`, is then authenticated. Once authentication is complete, regular LDAP operations can begin.

Searching the Directory

Searches are the most common LDAP operations. Applications can use complex search criteria to select and retrieve entries from the directory.

This section contains these topics:

- Program Flow for Search Operations
- Search Scope
- Filters
- Searching the Directory by Using the C API
- Searching the Directory by Using DBMS_LDAP

> **Note:** This release of the `DBMS_LDAP` API provides only synchronous search capability. This means that the caller of the search functions is blocked until the LDAP server returns the entire result set.

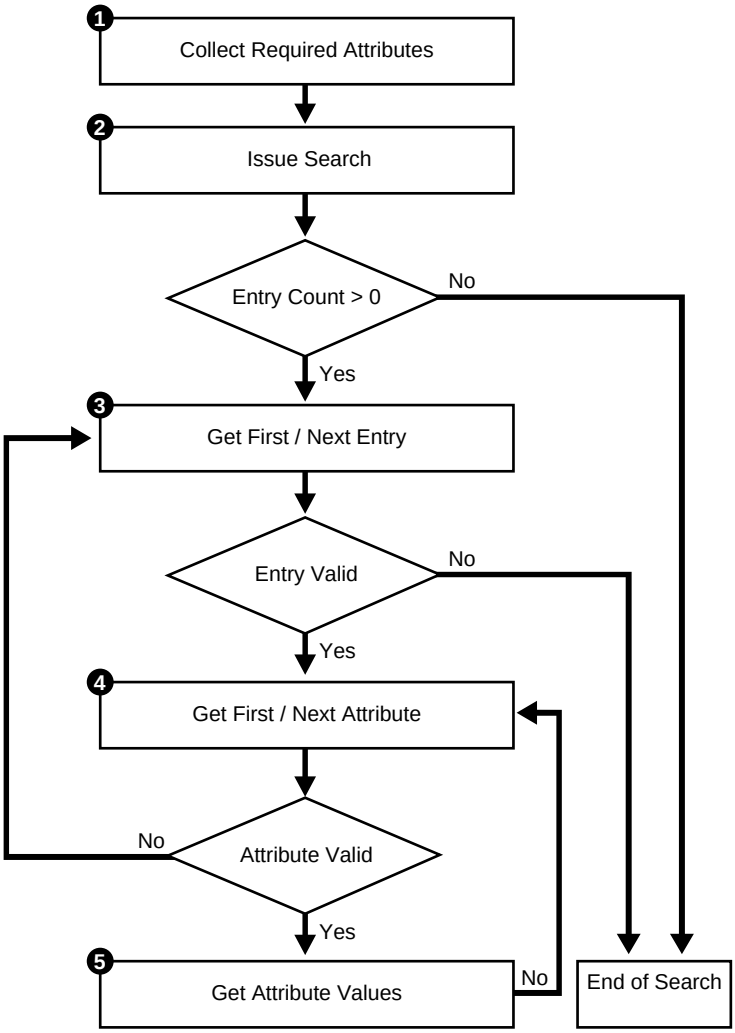
Program Flow for Search Operations

The programming required to initiate a typical search operation and retrieve results can be broken down into the following steps:

1. Decide what attributes must be returned; then place them into an array.
2. Initiate the search, using the scope options and filters of your choice.
3. Obtain an entry from result set.
4. Obtain an attribute from the entry obtained in step 3.
5. Obtain the values of the attributes obtained in step 4; then copy these values into local variables.
6. Repeat step 4 until all attributes of the entry are examined.
7. Repeat Step 3 until there are no more entries

Figure 2–4 on page 2-13 uses a flow chart to represent these steps.

Figure 2–4 Flow of Search-Related Operations



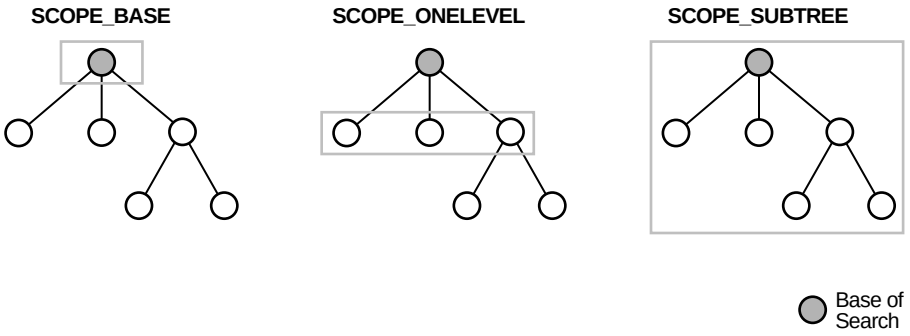
Search Scope

The scope of a search determines how many entries the directory server examines relative to the search base. You can choose one of the three options described in Table 2–5 and illustrated in Figure 2–5 on page 2-14.

Table 2–5 Options for search_s() or search_st() Functions

Option	Description
`SCOPE_BASE`	The directory server looks only for the entry corresponding to the search base.
`SCOPE_ONELEVEL`	The directory server confines its search to the entries that are the immediate children of the search base entry.
`SCOPE_SUBTREE`	The directory server looks at the search base entry and the entire subtree beneath it.

Figure 2–5 The Three Scope Options



In Figure 2–5, the search base is the shaded circle. The shaded rectangle identifies the entries that are searched.

Filters

A search filter is an expression that enables you to confine your search to certain types of entries. The search filter required by the `search_s()` and `search_st()` functions follows the string format defined in RFC 1960 of the Internet Engineering Task Force (IETF). As Table 2–6 shows, there are six kinds of search filters. These are entered in the format *`attribute operator value`*.

Table 2–6 Search Filters

Filter Type	Format	Example	Matches
Equality	(*att*=*value*)	(sn=Keaton)	Surnames exactly equal to Keaton.
Approximate	(*att*~=*value*)	(sn~=Ketan)	Surnames approximately equal to Ketan.
Substring	(attr=[*leading*]*[*any*]*[*trailing*]	(sn=*keaton*)	Surnames containing the string keaton.
		(sn=keaton*)	Surnames starting with keaton.
		(sn=*keaton)	Surnames ending with keaton.
		(sn=ke*at*on)	Surnames starting with ke, containing at and ending with on.
Greater than or equal	*attr*>=*value*	(sn>=Keaton)	Surnames lexicographically greater than or equal to Keaton.
Less than or equal	(*attr*<=*value*)	(sn<=Keaton)	Surnames lexicographically less than or equal to Keaton.
Presence	(*attr*=*)	(sn=*)	All entries having the sn attribute.

You can use boolean operators and prefix notation to combine these filters to form more complex filters. Table 2–7 on page 2-15 provides examples. In these examples, the

& character represents AND, the | character represents OR, and the ! character represents NOT.

Table 2–7 Boolean Operators

Filter Type	Format	Example	Matches
AND	(&(*filter1*)(*filter2*)). . .)	(&(sn=keaton)(objectclass=inetOrgPerson))	Entries with surname of Keaton and object class of InetOrgPerson.
OR	(|(*filter1*)(*filter2*)). . .)	(|(sn~=ketan)(cn=*keaton))	Entries with surname approximately equal to ketan or common name ending in keaton.
NOT	(!(*filter*))	(!(mail=*))	Entries without a mail attribute.

The complex filters in Table 2–7 can themselves be combined to create even more complex, nested filters.

Searching the Directory by Using the C API

The C function ldap_search_s() performs a synchronous search of the directory.

The syntax for ldap_search_s()looks like this:

```
int ldap_search_s
(
LDAP*ld,
char*base,
intscope,
char*filter,
intattrsonly,
LDAPMessage**res,
);
```

ldap_search_s works with several supporting functions to refine the search. The steps that follow show how all of these C functions fit into the program flow of a search operation. Chapter 8, "C API Reference", examines all of these functions in depth.

1. Decide what attributes must be returned; then place them into an array of strings. The array must be null terminated.
2. Initiate the search, using ldap_search_s(). Refine your search with scope options and filters.
3. Obtain an entry from the result set, using either the ldap_first_entry() function or the ldap_next_entry() function.
4. Obtain an attribute from the entry obtained in step 3. Use either the ldap_first_attribute() function or the ldap_next_attribute() function for this purpose.
5. Obtain all the values for the attribute obtained in step 4; then copy these values into local variables. Use the ldap_get_values() function or the ldap_get_values_len() function for this purpose.
6. Repeat step 4 until all attributes of the entry are examined.

7. Repeat step 3 until there are no more entries.

Table 2–8 Arguments for ldap_search_s()

Argument	Description
ld	A valid LDAP session handle
base	The DN of the search base.
scope	The breadth and depth of the DIT to be searched.
filter	The filter used to select entries of interest.
attrs	The attributes of interest in the entries returned.
attrso	If set to 1, only returns attributes.
res	This argument returns the search results.

Searching the Directory by Using DBMS_LDAP

You use the function DBMS_LDAP.search_s()to performs directory searches if you use the PL/SQL API.

Here is the syntax for DBMS_LDAP.search_s():

```
FUNCTION search_s
(
ld       IN  SESSION,
base     IN  VARCHAR2,
scope    IN  PLS_INTEGER,
filter   IN  VARCHAR2,
attrs    IN  STRING_COLLECTION,
attronly IN  PLS_INTEGER,
res      OUT MESSAGE
)
RETURN PLS_INTEGER;
```

The function takes the arguments listed and described in Table 2–9 on page 2-16.

Table 2–9 Arguments for DBMS_LDAP.search_s() and DBMS_LDAP.search_st()

Argument	Description
ld	A valid session handle
base	The DN of the base entry in the LDAP server where search should start
scope	The breadth and depth of the DIT that needs to be searched
filter	The filter used to select entries of interest
attrs	The attributes of interest in the entries returned
attronly	If set to 1, only returns the attributes
res	An OUT parameter that returns the result set for further processing

search_s works with several supporting functions to refine the search. The steps that follow show how all of these PL/SQL functions fit into the program flow of a search operation.

1. Decide what attributes need to be returned; then place them into the DBMS_LDAP.STRING_COLLECTION data-type.

2. Perform the search, using either `DBMS_LDAP.search_s()`or `DBMS_LDAP.search_st()`. Refine your search with scope options and filters.
3. Obtain an entry from the result set, using either`DBMS_LDAP.first_entry()` or `DBMS_LDAP.next_entry()`.
4. Obtain an attribute from the entry obtained in step 3. Use either `DBMS_LDAP.first_attribute()` or `DBMS_LDAP.next_attribute()` for this purpose.
5. Obtain all the values for the attribute obtained in step 4; then copy these values into local variables. Use either `DBMS_LDAP.get_values()` or `DBMS_LDAP.get_values_len()` for this purpose.
6. Repeat step 4 until all attributes of the entry are examined.
7. Repeat step 3 until there are no more entries.

Terminating the Session

This section contains these topics:

- Terminating the Session by Using the C API
- Terminating the Session by Using DBMS_LDAP

Terminating the Session by Using the C API

Once an LDAP session handle is obtained and all directory-related work is complete, the LDAP session must be destroyed. In the C API, the `ldap_unbind_s()` function is used for this purpose.

`ldap_unbind_s()` has this syntax:

```
int ldap_unbind_s
(
LDAP* ld
);
```

A successful call to `ldap_unbind_s()`closes the TCP/IP connection to the directory. It de-allocates system resources consumed by the LDAP session. Finally it returns the integer `LDAP_SUCCESS` to its callers. Once `ldap_unbind_s()`is invoked, no other LDAP operations are possible. A new session must be started with `ldap_init()`.

Terminating the Session by Using DBMS_LDAP

The `DBMS_LDAP.unbind_s()` function destroys an LDAP session if the PL/SQL API is used. `unbind_s` has the following syntax:

```
FUNCTION unbind_s (ld IN SESSION )  RETURN PLS_INTEGER;
```

`unbind_s` closes the TCP/IP connection to the directory. It de-allocates system resources consumed by the LDAP session. Finally it returns the integer `DBMS_LDAP.SUCCESS` to its callers. Once the `unbind_s` is invoked, no other LDAP operations are possible. A new session must be initiated with the `init` function.

3

Developing Applications with Oracle Extensions to the Standard APIs

This chapter presents the Oracle extensions to the LDAP APIs. It includes sample use cases.

This chapter contains these topics:

Using Oracle Extensions to the Standard APIs

The APIs that Oracle has added to the existing APIs fulfill these functions:

- User management

  Applications can set or retrieve various user properties
- Group management

  Applications can query group properties
- Realm management

  Applications can set or retrieve properties about identity management realms
- Server discovery management

  Applications can locate a directory server in the Domain Name System (DNS)
- SASL management

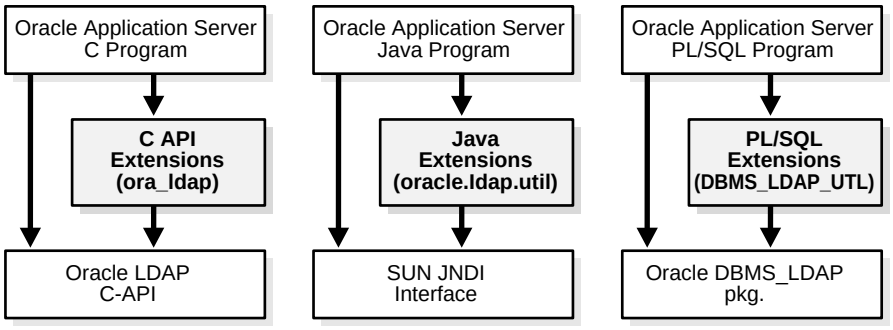
  Applications can authenticate to the directory using SASL Digest-MD5

The primary users of the Oracle extensions are backend applications that must perform LDAP lookups for users, groups, applications, or hosted companies. This section explains how these applications integrate these API extensions into their program logic. The section contains these topics:

- Using the API Extensions in PL/SQL
- Using the API Extensions in Java
- How the Standard APIs and The Oracle Extensions Are Installed

Figure 3–1 shows the placement of the API extensions in relation to existing APIs:

Figure 3–1 Oracle API Extensions

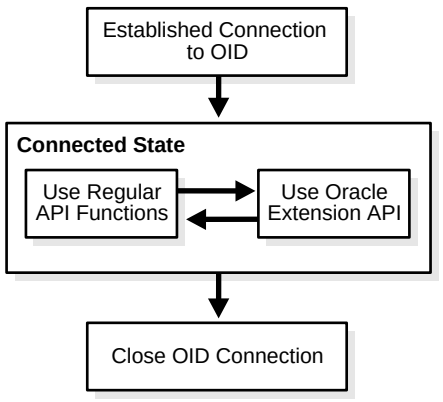


As Figure 3–1 shows, in the C, PL/SQL, and Java languages, the API extensions are layers that sit on top of existing APIs:

Applications must use the underlying APIs for such common tasks as establishing and closing connections and looking up directory entries not searchable with the API extensions.

Figure 3–2 shows what program flow looks like when the API extensions are used.

Figure 3–2 Programmatic Flow for API Extensions



As Figure 3–2 shows, an application first establishes a connection to Oracle Internet Directory. It can then use the standard API functions and the API extensions interchangeably.

Using the API Extensions in PL/SQL

Most of the extensions described in this chapter are helper functions. They access data about specific LDAP entities such as users, groups, realms, and applications. In many cases, these functions must pass a reference to one of these entities to the standard API functions. To do this, the API extensions use opaque data structures called handles. The steps that follow show an extension creating a user handle:

1. Establish an LDAP connection or get one from a pool of connections.
2. Create a user handle from user input. This could be a DN, a GUID, or a single sign-on user ID.
3. Authenticate the user with the LDAP connection handle, user handle, or credentials.
4. Free the user handle.
5. Close the LDAP connection, or return the connection back to the connection pool.

Using the API Extensions in Java

This section describes:

- The `oracle.java.util` package
- The `PropertySetCollection`, `PropertySet`, and `Property` classes

The oracle.java.util Package

In Java, LDAP entities—users, groups, realms, and applications—are modeled as Java objects instead of as handles. This modeling is done in the `oracle.java.util` package. All other utility functionality is modeled either as individual objects—as, for example, `GUID`—or as static member functions of a utility class.

For example, to authenticate a user, an application must follow these steps:

1. Create `oracle.ldap.util.user` object, given the user DN.
2. Create a `DirContext` JNDI object with all of the required properties, or get one from a pool of `DirContext` objects.
3. Invoke the User.authenticate function, passing in a reference to the `DirContext` object and the user credentials.
4. If the `DirContext` object was retrieved from a pool of existing `DirContext` objects, return it to that pool.

Unlike their C and PL/SQL counterparts, Java programmers do not have to explicitly free objects. The Java garbage collection mechanism performs this task.

PropertySetCollection, PropertySet, and Property Classes

Many of the methods in the user, subscriber, and group classes return a `PropertySetCollection` object. The object represents a collection of one or more LDAP entries. Each of these entries is represented by a `PropertySet` object, identified by a DN. A property set can contain attributes, each represented as a property. A property is a collection of one or more values for the particular attribute it represents. An example of the use of these classes follows:

```
PropertySetCollection psc = Util.getGroupMembership( ctx,
                                                     myuser,
                                                     null,
                                                     true );
```

```
     // for loop to go through each PropertySet
     for (int i = 0; i < psc.size(); i++ ) {

     PropertySet ps = psc.getPropertySet(i);

    // Print the DN of each PropertySet
    System.out.println("dn:  " + ps .getDN());

    // Get the values for the "objectclass" Property
    Property objectclass = ps.getProperty( "objectclass" );


    // for loop to go through each value of Property "objectclass"
    for (int j = 0; j< objectclass.size(); j++) {

        // Print each "objectclass" value
        System.out.println("objectclass:  " + objectclass.getValue(j));
    }
}
```

The entity `myuser` is a user object. The `psc` object contains all the nested groups that myuser belongs to. The code loops through the resulting entries and prints out all the object class values of each entry.

> **See Also:** "Java Sample Code" on page B-23 for more sample uses of the `PropertySetCollection`, `PropertySet`, and `Property` classes

How the Standard APIs and The Oracle Extensions Are Installed

Table 3–1 explains how the APIs and their extensions are installed.

Table 3–1 How the APIs are Installed

Language	Installation Method
Java API	Installed as part of the LDAP client installation. The file, `ldapjclnt10.jar`, is found at *`ORACLE_HOME`*`/jlib`.
PL/SQL API	Installed as part of the Oracle database server. Load it by using a script called `catldap.sql`, located at *`ORACLE_HOME`*`/rdbms/admin`.
C API	To build applications with the C API, include the header file located at *`ORACLE_HOME`*`/ldap/public/ldap.h`; then link dynamically to the library located at *`ORACLE_HOME`*`/lib/libclntsh.so.10.1`.

Creating an Application Identity in the Directory

Before an application can use the LDAP APIs and their extensions, it must establish an LDAP connection. Once it establishes a connection, it must have permission to perform operations. But neither task can be completed if the application lacks an identity in the directory.

Creating an Application Identity

Creating an application identity in the directory is relatively simple. Such an entry requires only two object classes: `orclApplicationEntity` and `top`. You can use

either Oracle Directory Manager or an LDIF file to create the entry. In LDIF notation, the entry looks like this:

```
dn: orclapplicationcommonname=application_name
changetype: add
objectclass:top
objectclass: orclApplicationEntity
userpassword: password
```

The value provided for `userpassword` is the value that the application uses to bind to the directory.

Assigning Privileges to an Application Identity

To learn about the privileges available to an application, see the chapter about delegating privileges for an Oracle technology deployment in *Oracle Internet Directory Administrator's Guide*. After identifying the right set of privileges, add the application entity DN to the appropriate directory groups. The link just provided explains how to perform this task using either Oracle Directory Manager or the `ldapmodify` command.

User Management Functionality

This section explains how the Java, PL/SQL, and C LDAP APIs are used to manage end users. It contains these topics:

- User Operations Performed by Directory-Enabled Applications
- User Management APIs
- User Authentication
- User Creation
- User Object Retrieval

User Operations Performed by Directory-Enabled Applications

Directory-enabled applications need to perform the following operations:

- Retrieve properties of user entries

  These properties are stored as attributes of the user entry itself—in the same way, for example, that a surname or a home address is stored.

- Retrieve extended user preferences

  These preferences apply to a user but are stored in a DIT different from the DIT containing user entries. Extended user preferences are either user properties common to all applications or user properties specific to an application. Those of the first type are stored in a common location in the Oracle Context. Those of the second type are stored in the application-specific DIT.

- Query the group membership of a user
- Authenticate a user given a simple name and credential

  Typically an application uses a fully qualified DN, GUID, or simple user name to identify a user. In a hosted environment, the application may use both a user name and a realm name for identification.

User Management APIs

This section looks at the user management features of the APIs.

Java API for User Management

As stated earlier, all user-related functionality is abstracted in a Java class called `oracle.ldap.util.User`. The process works like this:

1. Construct a `oracle.ldap.util.User` object based on a DN, GUID, or simple name.
2. Invoke `User.authenticate(DirContext, Credentials)` to authenticate the user if necessary.
3. Invoke `User.getProperties(DirContext)` to get the attributes of the user entry.
4. Invoke `User.getExtendedProperties(DirContext, PropCategory, PropType)` to get the extended properties of the user. `PropCategory` is either shared or application-specific. `PropType` is the object that represents the type of property desired. If `PropType` is null, all properties in a given category are retrieved.
5. Invoke `PropertyType.getDefinition(DirContext)` to get the metadata required to parse the properties returned in step 4.
6. Parse the extended properties and continue with application-specific logic. This parsing is also performed by application-specific logic.

C API for User Management

Oracle Internet Directory does not support the C API for user management.

PL/SQL API for User Management

The steps that follow show how the DBMS_LDAP_UTL package is used to create and use a handle that retrieves user properties from the directory.

1. Invoke `DBMS_LDAP_UTL.create_user_handle(user_hd, user_type, user_id)` to create a user handle from user input. The input can be a DN, a GUID, or a single sign-on user ID.
2. Invoke `DBMS_LDAP_UTL.set_user_handle_properties(user_hd, property_type, property)` to associate a realm with the user handle.
3. Invoke `DBMS_LDAP_UTL.get_user_properties(ld, user_handle, attrs, ptype, ret_pset_coll)` to place the attributes of a user entry into a result handle.
4. Invoke `DBMS_LDAP_UTL.get_property_names(pset, property_names)` and `DBMS_LDAP_UTL.get_property_values(pset, property_name, property_values)` to extract user attributes from the result handle that you obtained in step 3.

User Authentication

This section looks at the user authentication features of the APIs.

Java API for User Authentication

User authentication is a common LDAP operation that compares the credentials that a user provides at login with the user's credentials in the directory. Oracle Internet Directory supports the following:

- Arbitrary attributes can be used during authentication
- Appropriate password policy exceptions are returned by the authentication method. Note, however, that the password policy applies only to the `userpassword` attribute.

The following is a piece code that shows how the API is used to authenticate a user:

```
 // User user1 - is a valid User Object
        try
        {
                user1.authenticateUser(ctx,
User.CREDTYPE_PASSWD, "welcome");

                // or
                // user1.authenticateUser(ctx, <any
attribute>, <attribute value>);
        }
        catch (UtilException ue)
        {
                // Handle the password policy error
accordingly
                if (ue instanceof PasswordExpiredException)
                        // do something
                else if (ue instanceof GraceLoginException)
                        // do something
        }
```

PL/SQL API for User Authentication

Use `DBMS_LDAP_UTL.authenticate_user(session, user_handle, auth_type, cred, binary_cred)` to authenticate a user to the directory. This function compares the password provided by the user with the password attribute in the user's directory entry.

C API for User Authentication

Oracle Internet Directory does not support the C API for user authentication.

User Creation

This section looks at the user creation features of the APIs.

Java API for User Creation

The subscriber class uses the `createUser()` method to programmatically create users. The object classes required by a user entry are configurable through Oracle Delegated Administration Services. The `createUser()` method assumes that the client understands the requirement and supplies the values for the mandatory attributes during user creation. If the programmer does not supply the required information the server will return an error.

The following snippet of sample code demonstrates the usage.

```
// Subscriber sub is a valid Subscriber object
// DirContext ctx is a valid DirContext
```

```
// Create ModPropertySet object to define all the attributes and their values.
ModPropertySet mps = new ModPropertySet();
mps.addProperty(LDIF.ATTRIBUTE_CHANGE_TYPE_ADD,"cn", "Anika");
mps.addProperty(LDIF.ATTRIBUTE_CHANGE_TYPE_ADD,"sn", "Anika");
mps.addProperty(LDIF.ATTRIBUTE_CHANGE_TYPE_ADD,"mail",
"Anika@oracle.com");

// Create user by specifying the nickname and the ModPropertySet just defined
User newUser = sub.createUser( ctx, mps);

// Print the newly created user DN
System.out.println( newUser.getDN(ctx) );

// Perform other operations with this new user
```

PL/SQL API for User Creation

Oracle Internet Directory does not support the PL/SQL API for user creation.

C API for User Creation

Oracle Internet Directory does not support the PL/SQL API for user creation.

User Object Retrieval

This section describes user object retrieval features of the Java, PL/SQL, and C LDAP APIs.

Java API for User Object Retrieval

The subscriber class offers the `getUser()` method to replace the public constructors of the User class. A user object is returned based on the specified information.

The following is a piece of sample code demonstrating the usage:

```
// DirContext ctx is contains a valid directory connection with
sufficient privilege to perform the operations

// Creating RootOracleContext object
RootOracleContext roc = new RootOracleContext(ctx);

// Obtain a Subscriber object representing the default
subscriber
Subscriber sub = roc.getSubscriber(ctx,
Util.IDTYPE_DEFAULT, null, null);

// Obtain a User object representing the user whose
nickname is "Anika"
User user1 = sub.getUser(ctx, Util.IDTYPE_SIMPLE, "Anika",
null);
// Do work with this user

The getUser() method can retrieve users based on DN, GUID
and simple name.  A getUsers() method is also available to
perform a filtered search to return more than one user at a
time.  The returned object is an array of User objects.
For example,

// Obtain an array of User object where the user's nickname
starts with "Ani"
```

```
User[] userArr = sub.getUsers(ctx, Util.IDTYPE_SIMPLE,
"Ani", null);
// Do work with the User array
```

PL/SQL API for User Object Retrieval

Oracle Internet Directory does not support the PL/SQL API for user object retrieval.

C API for User Object Retrieval

Oracle Internet Directory does not support the C API for user object retrieval.

Group Management Functionality

This section describes the group management features of the Java, PL/SQL, and C LDAP APIs.

Groups are modeled in Oracle Internet Directory as a collection of distinguished names. Directory-enabled applications must access Oracle Internet Directory to obtain the properties of a group and to verify that a given user is a member of that group.

A group is typically identified by one of the following:

- A fully qualified LDAP distinguished name
- A global unique identifier
- A simple group name along with a subscriber name

Identity Management Realm Functionality

This section describes the identity management realm features of the Java, PL/SQL, and C LDAP APIs.

An identity management realm is an entity or organization that subscribes to the services offered in the Oracle product stack. Directory-enabled applications must access Oracle Internet Directory to obtain realm properties such as user search base or password policy.

A realm is typically identified by one of the following:

- A fully qualified LDAP distinguished name
- A global unique identifier
- A simple enterprise name

Realm Object Retrieval for the Java API

This section describes how the Java API can be used to retrieve objects in identity management realms.

The `RootOracleContext` class represents the root Oracle Context. Much of the information needed for identity management realm creation is stored within the root Oracle Context. The `RootOracleContext` class offers the `getSubscriber()` method. It replaces the public constructors of the subscriber class and returns an identity management realm object based on the specified information.

The following is a piece of sample code demonstrating the usage:

```
// DirContext ctx contains a valid directory
// connection with sufficient privilege to perform the
```

```
// operations

// Creating RootOracleContext object
RootOracleContext roc = new RootOracleContext(ctx);

// Obtain a Subscriber object representing the
// Subscriber with simple name "Oracle"
Subscriber sub = roc.getSubscriber(ctx,
Util.IDTYPE_SIMPLE, "Oracle", null);

// Do work with the Subscriber object
```

Server Discovery Functionality

Directory server discovery (DSD) enables automatic discovery of the Oracle directory server by directory clients. It enables deployments to manage the directory host name and port number information in the central DNS server. All directory clients perform a DNS query at runtime and connect to the directory server. Directory server location information is stored in a DNS service location record (SRV).

An SRV contains:

- The DNS name of the server providing LDAP service
- The port number of the corresponding port
- Any parameters that enable the client to choose an appropriate server from multiple servers

DSD also allows clients to discover the directory host name information from the `ldap.ora` file itself.

This section contains these topics:

See Also:

- "Discovering LDAP Services with DNS" by Michael P. Armijo at this URL:

  `http://www.ietf.org/`

- "A DNS RR for specifying the location of services (DNS SRV)", Internet RFC 2782 at the same URL.

Benefits of Oracle Internet Directory Discovery Interfaces

Typically, the LDAP host name and port information is provided statically in a file called `ldap.ora` which is located on the client in *`ORACLE_HOME`*`/network/admin`. For large deployments with many clients, this information becomes very cumbersome

to manage. For example, each time the host name or port number of a directory server is changed, the `ldap.ora` file on each client must be modified.

Directory server discovery eliminates the need to manage the host name and port number in the `ldap.ora` file. Because the host name information resides on one central DNS server, the information must be updated only once. All clients can then discover the new host name information dynamically from the DNS when they connect to it.

DSD provides a single interface to obtain directory server information without regard to the mechanism or standard used to obtain it. Currently, Oracle directory server information can be obtained either from DNS or from `ldap.ora` using a single interface.

Usage Model for Discovery Interfaces

The first step in discovering host name information is to create a discovery handle. A discovery handle specifies the source from which host name information will be discovered. In case of the Java API, the discovery handle is created by creating an instance of the `oracle.ldap.util.discovery.DiscoveryHelper` class.

```
DiscoveryHelper disco = new DiscoveryHelper(DiscoveryHelper.DNS_DISCOVER);
```

The argument `DiscoveryHelper.DNS_DISCOVER` specifies the source. In this case the source is DNS.

Each source may require some inputs to be specified for discovery of host name information. In the case of DNS these inputs are:

- domain name
- discover method
- SSL mode

Detailed explanation of these options is given in Determining Server Name and Port Number From DNS.

```
// Set the property for the DNS_DN
disco.setProperty(DiscoveryHelper.DNS_DN,"dc=us,dc=fiction,dc=com");
// Set the property for the DNS_DISCOVER_METHOD
disco.setProperty(DiscoveryHelper.DNS_DISCOVER_METHOD
                                  ,DiscoveryHelper.USE_INPUT_DN_METHOD);
// Set the property for the SSLMODE
disco.setProperty(DiscoveryHelper.SSLMODE,"0");
```

Now the information can be discovered.

```
// Call the discover method
disco.discover(reshdl);
```

The discovered information is returned in a result handle (`reshdl`). Now the results can be extracted from the result handle.

```
ArrayList result =
(ArrayList)reshdl.get(DiscoveryHelper.DIR_SERVERS);
if (result != null)
{
   if (result.size() == 0) return;
   System.out.println("The hostnames are :-");
   for (int i = 0; i< result.size();i++)
   {
      String host = (String)result.get(i);
```

```
System.out.println((i+1)+".
'"+host+"'");
   }
}
```

Determining Server Name and Port Number From DNS

Determining a host name and port number from a DNS lookup involves obtaining a domain and then searching for SRV resource records based on that domain. If there is more than one SRV resource record, they are sorted by weight and priority. The SRV resource records contain host names and port numbers required for connection. This information is retrieved from the resourcerecords and returned to the user.

There are three approaches for determining the domain name required for lookup:

- Mapping the distinguished name (DN) of the naming context
- Using the domain component of local machine
- Looking up the default SRV record in the DNS

Mapping the DN of the Naming Context

The first approach is to map the distinguished name (DN) of naming context into domain name using the algorithm given here.

The output domain name is initially empty. The DN is processed sequentially from right to left. An RDN is able to be converted if it meets the following conditions:

- It consists of a single attribute type and value
- The attribute type is `dc`
- The attribute value is non-null

If the RDN can be converted, then the attribute value is used as a domain name component (label).

The first such value becomes the rightmost, and the most significant, domain name component. Successive converted RDN values extend to the left. If an RDN cannot be converted, then processing stops. If the output domain name is empty when processing stops, then the DN cannot be converted into a domain name.

For the DN `cn=John Doe,ou=accounting,dc=example,dc=net`, the client converts the dc components into the DNS name `example.net`.

Search by Domain Component of Local Machine

Sometimes a DN cannot be mapped to a domain name. For example, the DN `o=Oracle IDC,Bangalore` cannot be mapped to a domain name. In this case, the second approach uses the domain component of the local machine on which the client is running. For example, if the client machine domain name is `mc1.acme.com`, the domain name for the lookup is `acme.com`.

Search by Default SRV Record in DNS

The third approach looks for a default SRV record in the DNS. This record points to the default server in the deployment. The domain component for this default record is `_default`.

Once the domain name has been determined, it is used to send a query to DNS. The DNS is queried for SRV records specified in Oracle Internet Directory-specific format. For example, if the domain name obtained is `example.net`, the query for non-SSL

LDAP servers is for SRV resource records having the owner name `_ldap._tcp._oid.example.net`.

It is possible that no SRV resource records are returned from the DNS. In such a case the DNS lookup is performed for the SRV resource records specified in standard format. For example, the owner name would be `_ldap._tcp.example.net`.

> **See Also:** The chapter about directory administration in *Oracle Internet Directory Administrator's Guide*

The result of the query is a set of SRV records. These records are then sorted and the host information is extracted from them. This information is then returned to the user.

> **Note:** The approaches mentioned here can also be tried in succession, stopping when the query lookup of DNS is successful. Try the approaches in the order as described in this section. DNS is queried only for SRV records in Oracle Internet Directory-specific format. If none of the approaches is successful, then all the approaches are tried again, but this time DNS is queried for SRV records in standard format.

Environment Variables for DNS Server Discovery

The following environment variables override default behavior for discovering a DNS server.

Table 3–2 Environment Variables for DNS Discovery

Environment Variable	Description
`ORA_LDAP_DNS`	IP address of the DNS server containing the SRV records. If the variable is not defined, then the DNS server address is obtained from the host machine.
`ORA_LDAP_DNSPORT`	Port number on which the DNS server listens for queries. If the variable is not defined, then the DNS server is assumed to be listening at standard port number `53`.
`ORA_LDAP_DOMAIN`	Domain of the host machine. If the variable is not defined, then the domain is obtained from the host machine itself.

Programming Interfaces for DNS Server Discovery

The programming interface provided is a single interface to discover directory server information without regard to the mechanism or standard used to obtain it. Information can be discovered from various sources. Each source can use its own mechanism to discover the information. For example, the LDAP host and port information can be discovered from the DNS acting as the source. Here DSD is used to discover host name information from the DNS.

> **See Also:** For detailed reference information and class descriptions, refer to the Javadoc located on the product CD.

Java APIs for Server Discovery

A new Java class, the public class, has been introduced:

```
public     class                oracle.ldap.util.discovery.DiscoveryHelper
```

This class provides a method for discovering specific information from the specified source.

Table 3–3 Methods for Directory Server Discovery

Method	Description
`discover`	Discovers the specific information from a given source
`setProperty`	Sets the properties required for discovery
`getProperty`	Accesses the value of properties

Two new methods are added to the existing Java class `oracle.ldap.util.jndi.ConnectionUtil`:

- getDefaultDirCtx: This overloaded function determines the host name and port information of non-SSL ldap servers by making an internal call to `oracle.ldap.util.discovery.DiscoveryHelper.discover()`.
- `getSSLDirCtx`: This overloaded function determines the host name and port information of SSL ldap servers by making an internal call to `oracle.ldap.util.discovery.DiscoveryHelper.discover()`.

Examples: Java API for Directory Server Discovery

The following is a sample Java program for directory server discovery:

```
import java.util.*;
import java.lang.*;
import oracle.ldap.util.discovery.*;
import oracle.ldap.util.jndi.*;

public class dsdtest
{
  public static void main(String s[]) throws Exception
  {
    HashMap reshdl = new HashMap();
    String result = new String();
    Object resultObj = new Object();
    DiscoveryHelper disco = new
DiscoveryHelper(DiscoveryHelper.DNS_DISCOVER);

// Set the property for the DNS_DN
disco.setProperty(DiscoveryHelper.DNS_DN,"dc=us,dc=fiction,dc=com")
;

// Set the property for the DNS_DISCOVER_METHOD
disco.setProperty(DiscoveryHelper.DNS_DISCOVER_METHOD
                                  ,DiscoveryHelper.USE_INPUT_DN_METHOD);

// Set the property for the SSLMODE
disco.setProperty(DiscoveryHelper.SSLMODE,"0");

// Call the discover method
int res=disco.discover(reshdl);
if (res!=0)
   System.out.println("Error Code returned by the discover method is :"+res) ;

// Print the results
printReshdl(reshdl);
}
```

```
public static void printReshdl(HashMap reshdl)
{
  ArrayList result = (ArrayList)reshdl.get(DiscoveryHelper.DIR_SERVERS);

if (result != null)
{
  if (result.size() == 0) return;
  System.out.println("The hostnames are :-");
  for (int i = 0; i< result.size();i++)
  {
      String host = (String)result.get(i);
      System.out.println((i+1)+".
'"+host+"'");
   }
  }
 }
}
```

SASL Authentication Functionality

Oracle Internet Directory supports two mechanisms for SASL-based authentication. This section describes the two methods. It contains these topics:

- SASL Authentication by Using the DIGEST-MD5 Mechanism
- SASL Authentication by Using External Mechanism

SASL Authentication by Using the DIGEST-MD5 Mechanism

SASL Digest-MD5 authentication is the required authentication mechanism for LDAP Version 3 servers (RFC 2829). LDAP Version 2 does not support Digest-MD5.

The Digest-MD5 mechanism is described in RFC 2831 of the Internet Engineering Task Force. It is based on the HTTP Digest Authentication (RFC 2617).

> **See Also:** Internet Engineering Task Force Web site:
>
> `http://www.ietf.org`

This section contains these topics:

- Steps Involved in SASL Authentication by Using DIGEST-MD5
- JAVA APIs for SASL Authentication by Using DIGEST-MD5
- C APIs for SASL authentication using DIGEST-MD5
- SASL Authentication by Using External Mechanism

Steps Involved in SASL Authentication by Using DIGEST-MD5

SASL Digest-MD5 authenticates a user as follows:

1. The directory server sends data that includes various authentication options that it supports and a special token to the LDAP client.
2. The client responds by sending an encrypted response that indicates the authentication options that it has selected. The response is encrypted in such a way that proves that the client knows its password.
3. The directory server then decrypts and verifies the client's response.

To use the Digest-MD5 authentication mechanism, you can use either the Java API or the C API to set up the authentication.

JAVA APIs for SASL Authentication by Using DIGEST-MD5

When using JNDI to create a SASL connection, you must set these `javax.naming.Context` properties:

- `Context.SECURITY_AUTHENTICATION = "DIGEST-MD5"`
- `Context.SECURITY_PRINCIPAL`

The latter sets the principal name. This name is a server-specific format. It can be either of the following:

- The DN—that is, `dn:`—followed by the fully qualified DN of the entity being authenticated
- The string `u:` followed by the user identifier.

  The Oracle directory server accepts just a fully qualified DN such as `cn=user,ou=my department,o=my company`.

> **Note:** The SASL DN must be normalized before it is passed to the C or Java API that calls the SASL bind. To generate SASL verifiers, Oracle Internet Directory supports only normalized DNs.

See Also:

- "Authenticating to the Directory" on page 8-10
- "C API Usage for SASL-Based DIGEST-MD5 Authentication" on page 8-43
- JNDI:

  `http://java.sun.com/products/jndi/`

SASL Authentication by Using External Mechanism

The following is from section 7.4 of RFC 2222 of the Internet Engineering Task Force.

The mechanism name associated with external authentication is "EXTERNAL". The client sends an initial response with the authorization identity. The server uses information, external to SASL, to determine whether the client is authorized to authenticate as the authorization identity. If the client is so authorized, the server indicates successful completion of the authentication exchange; otherwise the server indicates failure.

The system providing this external information may be, for example, IPsec or SSL/TLS.

If the client sends the empty string as the authorization identity (thus requesting the authorization identity be derived from the client's authentication credentials), the authorization identity is to be derived from authentication credentials that exist in the system which is providing the external authentication.

Oracle Internet Directory provides the SASL external mechanism over an SSL mutual connection. The authorization identity (DN) is derived from the client certificate during the SSL network negotiation.

Proxying on Behalf of End Users

Often applications must perform operations that require impersonating an end user. An application may, for example, want to retrieve resource access descriptors for an end user. (Resource access descriptors are discussed in the concepts chapter of *Oracle Internet Directory Administrator's Guide*.)

A proxy switch occurs at run time on the JNDI context. An LDAP v3 feature, proxying can only be performed using `InitialLdapContext`, a subclass of `InitialDirContext`. If you use the Oracle extension `oracle.ldap.util.jndi.ConnectionUtil` to establish a connection (the example following), `InitialLdapContext` is always returned. If you use JNDI to establish the connection, make sure that it returns `InitialLdapContext`.

To perform the proxy switch to an end user, the user DN must be available. To learn how to obtain the DN, see the sample implementation of the `oracle.ldap.util.User` class at this URL:

`http://www.oracle.com/technology/sample_code/id_mgmt`

This code shows how the proxy switch occurs:

```
import oracle.ldap.util.jndi.*;
import javax.naming.directory.*;
import javax.naming.ldap.*;
import javax.naming.*;

public static void main(String args[])
{
  try{
       InitialLdapContext appCtx=ConnectionUtil.getDefaultDirCtx(args[0], // host
                                                                 args[1], // port
                                                                 args[2], // DN
                                                                 args[3]; // pass)
         // Do work as application
         // . . .
         String userDN=null;
         // assuming userDN has the end user DN value
         // Now switch to end user
         ctx.addToEnvironment(Context.SECURITY_PRINCIPAL, userDN);
         ctx.addToEnvironment("java.naming.security.credentials", "");
         Control ctls[] = {
           new ProxyControl()
         };
         ((LdapContext)ctx).reconnect(ctls);
         // Do work on behalf of end user
         // . . .
       }
    catch(NamingException ne)
      {
       // javax.naming.NamingException is thrown when an error occurs
      }
}
```

The `ProxyControl` class in the code immediately preceding implements a `javax.naming.ldap.Control`. To learn more about LDAP controls, see the section about supported controls in the schema appendix of *Oracle Internet Directory Administrator's Guide*. Here is an example of what the `ProxyControl` class might look like:

```
import javax.naming.*;
import javax.naming.ldap.Control;
import java.lang.*;

public class ProxyControl implements Control {

   public byte[] getEncodedValue() {
        return null;
   }

   public String getID() {
      return "2.16.840.1.113894.1.8.1";
   }

   public boolean isCritical() {
      return false;
   }
}
```

Creating Dynamic Password Verifiers

You can modify standard APIs to generate application passwords dynamically—that is, when users log in to an application. This feature has been designed to meet the needs of applications that provide parameters for password verifiers only at runtime.

This section contains the following topics:

Request Control for Dynamic Password Verifiers

Creating a password verifier dynamically involves modifying the LDAP authentication APIs `ldap_search` or `ldap_modify` to include parameters for password verifiers. An LDAP control called `DynamicVerifierRequestControl` is the mechanism for transmitting these parameters. It takes the place of the password verifier profile used to create password verifiers statically. Nevertheless, dynamic verifiers, like static verifiers, require that the directory attributes `orclrevpwd` (synchronized case) and `orclunsyncrevpwd` (unsynchronized case) be present and that these attributes be populated.

Note that the `orclpwdencryptionenable` attribute of the password policy entry in the user's realm must be set to `1` if `orclrevpwd` is to be generated. If you fail to set this attribute, an exception is thrown when the user tries to authenticate. To generate `orclunsyncrevpwd`, you must add the crypto type 3DES to the entry `cn=defaultSharedPINProfileEntry,cn=common,cn=products,cn=oracle context`.

Syntax for DynamicVerifierRequestControl

The request control looks like this:

```
DynamicVerifierRequestControl
controlOid:  2.16.840.1.113894.1.8.14
criticality:  FALSE
controlValue: an OCTET STRING whose value is the BER encoding of the following
type:

ControlValue ::= SEQUENCE {

                 version [0]
                 crypto  [1] CHOICE OPTIONAL {
                    SASL/MD5  [0] LDAPString,
                    SyncML1.0 [1] LDAPString,
                    SyncML1.1 [2] LDAPString,
                    CRAM-MD5  [3] LDAPString },
                 username [1] OPTIONAL LDAPString,
                 realm    [2] OPTIONAL LDAPString,
                 nonce    [3] OPTIONAL LDAPString,
                }
```

Note that the parameters in the control structure must be passed in the order in which they appear. Table 3–4 defines these parameters.

Table 3–4 Parameters in DynamicVerifierRequestControl

Parameter	Description
`controlOID`	The string that uniquely identifies the control structure.
`crypto`	The hashing algorithm. Choose one of the four identified in the control structure.
`username`	The distinguished name (DN) of the user. This value must always be included.
`realm`	A randomly chosen realm. It may be the identity management realm that the user belongs to. It may even be an application realm. Required only by the SASL/MD5 algorithm.
`nonce`	An arbitrary, randomly chosen value. Required by SYNCML1.0 and SYNCML1.1.

Parameters Required by the Hashing Algorithms

Table 3–5 lists the four hashing algorithms that are used to create dynamic password verifiers. The table also lists the parameters that each algorithm uses as building blocks. Note that, although all algorithms use the user name and password parameters, they differ in their use of the `realm` and `nonce` parameters.

Table 3–5 Parameters Required by the Hashing Algorithms

Algorithm	Parameters Required
SASL/MD5	`username`, `realm`, `password`
SYNCML1.0	`username`, `password`, `nonce`
SYNCML1.1	`username`, `password`, `nonce`
CRAM-MD5	`username`, `password`

Configuring the Authentication APIs

Applications that require password verifiers to be generated dynamically must include `DynamicVerifierRequestControl` in their authentication APIs. Either `ldap_`

`search` or `ldap_compare` must incorporate the `controlOID` and the control values as parameters. They must BER-encode the control values as shown in "Syntax for DynamicVerifierRequestControl"; then they must send both `controlOID` and the control values to the directory server.

Parameters Passed If ldap_search Is Used

If you want the application to authenticate the user, use `ldap_search` to pass the control structure. If `ldap_search` is used, the directory passes the password verifier that it creates to the client.

`ldap_search` must include the DN of the user, the `controlOID`, and the control values. If the user's password is a single sign-on password, the attribute passed is `authpassword`. If the password is a numeric pin or another type of unsynchronized password, the attribute passed is `orclpasswordverifier;orclcommonpin`.

Parameters Passed If ldap_compare Is Used

If you want Oracle Internet Directory to authenticate the user, use `ldap_compare` to pass the control structure. In this case, the directory retains the verifier and authenticates the user itself.

Like `ldap_search`, `ldap_compare` must include the DN of the user, the `controlOID`, the control values, and the user's password attribute. For `ldap_compare`, the password attribute is `orclpasswordverifier;orclcommonpin` (unsynchronized case).

Response Control for Dynamic Password Verifiers

When it encounters an error, the directory sends the LDAP control `DynamicVerifierResponseControl` to the client. This response control contains the error code. To learn about the error codes that the response control sends, see the troubleshooting chapter in *Oracle Internet Directory Administrator's Guide*.

Obtaining Privileges for the Dynamic Verifier Framework

If you want the directory to create password verifiers dynamically, you must add your application identity to the VerifierServices group of directory administrators. If you fail to perform this task, the directory returns an `LDAP_INSUFFICIENT_ACCESS` error.

Dependencies and Limitations for the PL/SQ LDAP API

The PL/SQL LDAP API for this release has the following limitations:

- The LDAP session handles obtained from the API are valid only for the duration of the database session. The LDAP session handles cannot be written to a table and reused in other database sessions.
- Only synchronous versions of LDAP API functions are supported in this release.

  The PL/SQL LDAP API requires a database connection to work. It cannot be used in client-side PL/SQL engines (like Oracle Forms) without a valid database connection.

4

Developing Provisioning-Integrated Applications

This chapter explains how to develop applications that can use the Oracle Directory Provisioning Integration Service, a component of Oracle Directory Integration and Provisioning. These applications can be either legacy or third-party applications that are based on the Oracle platform.

This chapter contains these topics:

> **See Also:** The chapter on the Oracle Directory Provisioning Integration Service in *Oracle Identity Management Integration Guide*

Introduction to the Oracle Directory Provisioning Integration Service

A big challenge in directory administration is managing provisioning information for the myriad accounts and applications that each user may need. For example, adding a user to an information system typically requires a substantial amount of application provisioning. It can include setting up an e-mail account, which in turn has specific settings for a mail quota, some default folders, and perhaps some distribution lists. If there are other connectivity applications that the user needs, then managing that user's accounts and personal profile can be overwhelming for a large enterprise. To meet this challenge, the Oracle Directory Provisioning Integration Service provides a platform for integrating applications. It enables you to add a user seamlessly to many key systems in just one step.

The Oracle Directory Provisioning Integration Service serves as a passthrough for user account information. Rather than provisioning a user with each individual application, you simply register applications with the provisioning service. This enables them to send provisioning information directly to Oracle Internet Directory and receive information from it. Users can then be provisioned at once for a default set of integrated applications. In this way, the Oracle Directory Provisioning Integration Service eliminates redundant processing for each individual application.

In addition to a default set of provisioning events defined during installation, Oracle Internet Directory can define new events and propagate them appropriately to applications that subscribe to those events. The ability to both send and receive these provisioning events provides for seamless management of user accounts.

Developing Provisioning-Integrated Applications

Applications integrated with the Oracle Directory Provisioning Integration Service can be either legacy or third-party applications based on the Oracle platform. Once it has registered with Oracle Internet Directory, an application can send and receive provisioning information to and from the directory.

To integrate an application with the directory provisioning integration service, you follow these general steps, each of which is explained more fully later in this chapter:

- Register the application in Oracle Internet Directory.
- Identify the identity management realm under which events are to be propagated or to be applied.
- Determine whether the application needs to receive events, send events, or both.
- List the events that need to be sent or received.
- List attributes of interest that an event should contain.
- Assign proper privileges to the application identity in the identity management realm. This enables the application to read events from Oracle Internet Directory and propagate events to it.
- Determine the interface name, interface type, and interface connection. This is required by the provisioning server to propagate events to the application and consume events from it.
- Determine the other provisioning scheduling interval, maximum number of events per schedule, and so on.
- Implement the interface specifications inside the application.
- Create the provisioning profile in Oracle Internet Directory so that event propagation can start. Create this profile by using the provisioning subscription tool (`oidprovtool`).

The section that follows uses a sample application to show how these steps are implemented.

Example of a Provisioning-Integrated Application

This example of a provisioning-integrated application is called Employee Self Service Application (ESSA). In this discussion, the terms "user" and "identity" are used interchangeably.

Requirements of the Employee Self Service Application

This application requires that its entire user base be managed from Oracle Internet Directory. The application administrator creates, modifies, and deletes identities in Oracle Internet Directory. The identity information is propagated to the application as an event, namely, `IDENTITY_ADD`.

Although the application creates the identity as user data, this is not sufficient to authorize the employee to access the application. The presence of the identity in Oracle Internet Directory only facilitates a global login. The application must discover whether a particular identity is authorized to access the application. This is achieved by subscribing the identity for that application, a task that the application administrator can do. This subscribing triggers another event from Oracle Internet Directory to the application—namely, `SUBSCRIPTION_ADD`—indicating that the identity has now been subscribed in Oracle Internet Directory to use that application.

The application can then query the directory to check whether a particular user is present in the application subscription lists before allowing the user access to the application.

In this example, the events for this application are received from Oracle Internet Directory. The application itself does not send any events to the directory. It could, however, also send events to Oracle Internet Directory. To do this, the application identity needs more directory privileges for the various operations that it wants to perform on the directory. This is explained in "Determining Provisioning Mode for the Employee Self Service Application" on page 4-4.

The steps are as follows:

1. A user is added in Oracle Internet Directory through either the Oracle Internet Directory Self-Service Console or some other means such as synchronization from third party sources or through command-line tools. The user information must be placed in the appropriate identity management realm.
2. The `IDENTITY_ADD` event is propagated from Oracle Internet Directory to the application. This assumes that the application subscribed to the `IDENTITY_ADD` event during creation of the provisioning subscription profile.
3. On receiving the event, the application adds this identity to its database. In this example, however, this does not mean that the user is authorized to access the application. An additional event is required to subscribe the user as an authorized user of that application.
4. In Oracle Internet Directory, the user is subscribed to the application by using Oracle Delegated Administration Services.
5. The `SUBSCRIPTION_ADD` event is propagated from Oracle Internet Directory to the application. This assumes that the application subscribed to the `SUBSCRIPTION_ADD` event during creation of the provisioning subscription profile.
6. On receiving this event, the application updates the identity record in its database indicating that this is also an authorized user.

Registering the Employee Self Service Application in Oracle Internet Directory

The application must register itself as an application entity with its own identity entry in Oracle Internet Directory. You can decide which realm to create the application identity in, as long as that realm is a well-known location in the DIT. To create the necessary DIT elements in Oracle Internet Directory, you must follow a template described in this chapter.

The Oracle Context of the identity management realm has a container for the various application footprints. That container is:
`cn=products,cn=oraclecontext,`*`identity_management_realm_DN`*.

If the application is meant for only one realm, then Oracle Corporation recommends that you create the application identity DN in this form:
`orclApplicationName=`*`application_ name`*`,cn=`*`application_type`*`,cn=products,cn=oraclecontext,`*`identity_management_realm_DN`*.
The `cn=`*`application_type`* element is called the application container.

If the application is meant for multiple realms, you can create the application identity in the root Oracle Context, namely, `cn=products,cn=oraclecontext`.

In this example, the location and the content of the entry are as follows:

```
dn: \
orclApplicationCommonName=ESSA,cn=demoApps,cn=Products,cn=OracleContext,o=ACME,
dc=com
orclapplicationcommonname: ESSA
orclappfullname: Employee Self Service Application
userpassword: welcome123
description: This is an sample application for demonstration.
orclaci: access to entry by group="cn=odisgroup,cn=odi,cn=oracle internet direct
ory" (proxy)
objectclass: orclApplicationEntity
```

In this example, the application type or application container is `demoApps`. The application name is `ESSA`.

All directory operations must be done on the behalf of the application by the provisioning server. Because the server does not have privileges to send or consume events under the domain, it must process events by impersonating the application identity. This, in turn, requires that the server be given the proxy privilege. In this example, it is assumed that the application identity already has the necessary privileges.

Identifying the Management Context for the Employee Self Service Application

All identity management realms are generally present under the identity management realm base in the root Oracle Context. The application must be provisioned for the appropriate realm—that is, proper privileges must be assigned to this application identity so that it can administer its information under this realm. In this example, let us assume that the appropriate realm is `o=ACME,dc=com`.

Determining Provisioning Mode for the Employee Self Service Application

You must decide whether the application only receives events or whether it also sends them to Oracle Internet Directory. The mode can be one of the following:

- `INBOUND`: from the application to Oracle Internet Directory
- `OUTBOUND`: from Oracle Internet Directory to the application
- `BOTH`

The default mode is `OUTBOUND`.

In this example, because the application is interested in only receiving events from Oracle Internet Directory, we specify the events as `OUTBOUND` only.

Determining Events for the Employee Self Service Application

During installation, a fixed set of events is predefined. You can define new events at runtime, but they can be propagated in the outbound mode only. The Oracle Directory Provisioning Integration Service can process only a fixed set of predefined events for the inbound mode.

In this example, we do not need to define any new events. The following events in Oracle Internet Directory must be propagated to our sample application:

- Identity creation (`IDENTITY_ADD`)
- Identity modification (`IDENTITY_MODIFY`)
- Identity employee deletion (`IDENTITY_DELETE`)
- Identity subscription addition (`SUBSCRIPTION_ADD`)

- Identity subscription modification (SUBSCRIPTION_MODIFY)
- Identity subscription deletion (SUBSCRIPTION_DELETE)

Provisioning the Employee Self Service Application for an Identity Management Realm

This is the most important step. It involves assigning the proper privileges to the application identity in the identity management realm. These privileges enable the application to read and apply the various events from Oracle Internet Directory and to send change events to Oracle Internet Directory. Inbound events, which result in modifying Oracle Internet Directory, require more privileges.

Generally, predefined groups are created when the identity management realm is created. The groups have different privileges as described in this section.

The following template describes all the appropriate ACLs required for an application to send or receive provisioning events.

The application identity must be added to the appropriate group, but this, in turn, depends on the privileges it requires. For example, if an application is interested only in receiving events from Oracle Internet Directory, then it does not need to be added to groups that can create or modify entries in this realm.

The template accepts a few variables. Once the variables are instantiated, the template becomes a proper LDIF file that can be executed against Oracle Internet Directory. You can adjust the variables according to the needs of your deployment.

In this example, the identity management realm is o=ACME,dc=com. The template of the LDIF file looks like this:

```
# This creates The Application Identity subtree
#
# The following variables are used :
# (Some of them are OPTIONAL where the values oidprov tool can get  default
# values if not supplied.)
#
# %s_IdentityRealm% : Identity Realm DN:
# (MANDATORY: This is the domain in which all the related users and groups are
# present. If Default Identity Realm needs to be used,
# it can be queried in a directory install.
# This value is stored in the root Oracle Context of the directory.
# The value  is  stored  in the 'orcldefaultsubscriber'
# attribute in 'dn: cn=Common,cn=Products,cn=OracleContext')
#  %s_AppType% : Application Type (e.g EBusiness)
#                       (MANDATORY : Name of the suite  )
#  %s_AppName% : Application Name (e.g HRMS,Financials,Manufacturing)
#                       (MANDATORY: Name of the Application in the suite.)
#  %s_SvcType% : Service Type (e.g Ebusiness)
#                       (MANDATORY : Alias for name of suite.
#                                    This value can be be same as %s_AppType%)
#  %s_SvcName% : Service Name (e.g HRMS,Financials,Manufacturing)
#                       (MANDATORY : Alias for name of Application.
#                                    This value can be same as %s_AppName%)
#  %s_AppURL% : Application URL if any. (set it to 'NULL' if there is nothing.)
#
# Apart from these variables this LDIF templates would also need the following
# information to load this data to Oracle Internet Directory:
#
#  LDAP_HOST :  directory server hostname
#  LDAP_PORT :  directory server port number
#  BINDDN    :  cn=orcladmin
```

```
#  BINDPASSWD:  Password for orcladmin
#
# After replacing the variables in the template this data can be loaded into the
# directory by running the following command:
#  ldapmodify -h %LDAP_HOST% -p %LDAP_PORT% -D %BINDDN% \
#  -w %BINDPWD% -f <this_template_file_name>
#
#

# First we create the Application container. This needs to be created just once
# initially. If this container is existing  because some application was
# already created using this template, please remove this entry
# from the template/LDIF file.

dn: cn=%s_AppType%,cn=Products,cn=OracleContext,%s_IdentityRealm%
changetype: add
cn: %s_AppType%
objectclass: orclContainer

# The application identity needs to created next. This is under the Applications
# container. This object is of type "orclApplicationEntity"

dn: orclApplicationCommonName=%s_AppName%,cn=%s_
AppType%,cn=Products,cn=OracleContext,
 %s_IdentityRealm%
changetype: add
orclapplicationcommonname: %s_AppName%
orclaci: access to entry by group="cn=odisgroup,cn=odi,cn=oracle internet
directory"
 (add,browse,delete,proxy)
objectclass: orclApplicationEntity

# The following ACLs are for giving privileges to the application entities for
# adding/modifying/deleting users in the relevant realm.

# All members of the group represented by this DN are allowed to create users
# in the relevant realm:

dn: cn=OracleDASCreateUser,cn=Groups,cn=OracleContext,%s_IdentityRealm%
changetype: modify
add: uniquemember
uniquemember: orclApplicationCommonName=%s_AppName%,cn=%s_
AppType%,cn=Products,cn=OracleContext,
 %s_IdentityRealm%

# All members of the group represented by this DN are allowed to delete users in
# the relevant realm:

dn: cn=OracleDASDeleteUser,cn=Groups,cn=OracleContext,%s_IdentityRealm%
changetype: modify
add: uniquemember
uniquemember: orclApplicationCommonName=%s_AppName%,cn=%s_
AppType%,cn=Products,cn=OracleContext,
 %s_IdentityRealm%

# All members of the group represented by this DN are allowed to edit users in the
# relevant realm:

dn: cn=OracleDASEditUser,cn=Groups,cn=OracleContext,%s_IdentityRealm%
changetype: modify
```

```
add: uniquemember
uniquemember: orclApplicationCommonName=%s_AppName%,cn=%s_
AppType%,cn=Products,cn=OracleContext,
 %s_IdentityRealm%

# All members of the group represented by this DN are allowed to create groups in
# the relevant realm:

dn: cn=OracleDASCreateGroup,cn=Groups,cn=OracleContext,%s_IdentityRealm%
changetype: modify
add: uniquemember
uniquemember: orclApplicationCommonName=%s_AppName%,cn=%s_
AppType%,cn=Products,cn=OracleContext,
 %s_IdentityRealm%

# All members of the group represented by this DN are allowed to delete groups in
# the relevant realm:

dn: cn=OracleDASDeleteGroup,cn=Groups,cn=OracleContext,%s_IdentityRealm%
changetype: modify
add: uniquemember
uniquemember: orclApplicationCommonName=%s_AppName%,cn=%s_
AppType%,cn=Products,cn=OracleContext,
 %s_IdentityRealm%

# All members of the group represented by this DN are allowed to edit groups in
# the relevant realm:

dn: cn=OracleDASEditGroup,cn=Groups,cn=OracleContext,%s_IdentityRealm%
changetype: modify
add: uniquemember
uniquemember: orclApplicationCommonName=%s_AppName%,cn=%s_
AppType%,cn=Products,cn=OracleContext,
 %s_IdentityRealm%

# The container is being created to hold the various subscription lists of the
# application for this realm. This container will hold lots of subscription
# information and resides just under the application identity.

dn: cn=subscriptions,orclApplicationCommonName=%s_AppName%,cn=%s_
AppType%,cn=Products,
 cn=OracleContext,%s_IdentityRealm%
changetype: add
cn: subscriptions
objectclass: orclContainer

# The following is the group that will hold administrators DNs for managing
# subscription lists for this application. The application identity should also be
# in this list and will be added here.

dn: cn=Subscription_Admins,cn=Subscriptions,orclApplicationCommonName=%s_AppName%,
cn=%s_AppType%,cn=products,cn=OracleContext,%s_IdentityRealm%
changetype: add
cn: Subscription_Admins
uniquemember: orclApplicationCommonName=%s_AppName%,cn=%s_
AppType%,cn=Products,cn=OracleContext,
 %s_IdentityRealm%
objectclass: groupOfUniqueNames
objectclass: orclACPGroup
objectclass: orclprivilegegroup
```

```
# The following is the group that will hold DNs of users who can just view the
# subscription lists for this application. The application identity should also be
# in this list and will be added here.

dn: cn=Subscription_Viewers,cn=Subscriptions,orclApplicationCommonName=%s_
AppName%,
cn=%s_AppType%,cn=products,cn=OracleContext,%s_IdentityRealm%
changetype: add
cn: Subscription_Viewers
uniquemember: orclApplicationCommonName=%s_AppName%,cn=%s_
AppType%,cn=Products,cn=OracleContext,
%s_IdentityRealm%
objectclass: groupOfUniqueNames
objectclass: orclACPGroup
objectclass: orclprivilegegroup

# The following is just a container for the actual subscription lists.

dn: cn=subscription_data,cn=subscriptions,orclApplicationCommonName=%s_AppName%,
 cn=%s_AppType%,cn=Products,cn=OracleContext,%s_IdentityRealm%
changetype: add
cn: subscription_data
objectclass: orclContainer

# The following is a sample subscription list. We are calling it "cn=ACCOUNTS"
# since it signifies accounts in the application.

dn: cn=ACCOUNTS,cn=subscription_
data,cn=subscriptions,orclApplicationCommonName=%s_AppName%,cn=%s_
AppType%,cn=Products,cn=OracleContext,%s_IdentityRealm%
changetype: add
cn: cn=ACCOUNTS
uniquemember: cn=orcladmin
objectclass: groupOfUniqueNames
objectclass: orclGroup

# The following is a container for the service instance entries in the Root Oracle
# Context. An application publishes itself as a service by creating
# a service instance entry under this container. These service
# instance entries are created outside any realm and in the root Oracle Context.

dn: cn=%s_SvcType%,cn=Services,cn=OracleContext
changetype: add
cn: %s_SvcType%
objectclass: orclContainer

# The following is a container for the service instance entries in the Root Oracle
# Context for that service type.

dn: cn=ServiceInstances,cn=%s_SvcType%,cn=Services,cn=OracleContext
changetype: add
cn: ServiceInstances
objectclass: orclContainer

# The following is a service instance entry. An application publishes itself as a
# service by creating this service instance.

dn: cn=%s_SvcName,cn=ServiceInstances,%,cn=%s_
SvcType%,cn=Services,cn=OracleContext
```

```
changetype: add
cn: %s_SvcName%
orclServiceType: %s_SvcType%
presentationAddress: %s_AppURL%
objectclass: orclServiceInstance

# The following is a container for service instance reference entry that resides
# in the relevant realm.

dn: cn=%s_SvcType%,cn=Services,cn=OracleContext,%s_IdentityRealm%
changetype: add
cn: %s_SvcType%
objectclass: orclContainer

# It is a reference entry which actually points to the actual service instance
# entry as well as to the subscription list container for the application.

dn: cn=%s_SvcName%,cn=%s_SvcType%,cn=Services,cn=OracleContext,%s_IdentityRealm%
changetype: add
cn: %s_SvcName%
description: Link To the Actual Subscription Location for the Application and the
actual Service instance.
orclServiceInstanceLocation: cn=%s_SvcName%,cn=%s_
SvcType%,cn=Services,cn=OracleContext
orclServiceSubscriptionLocation: cn=subscription_data,cn=subscriptions,
 orclApplicationCommonName=%s_AppName%,cn=%s_
AppType%,cn=Products,cn=OracleContext,
 %s_IdentityRealm%
objectclass: orclServiceInstanceReference


# This LDIF operation gives appropriate privileges to the subscription admin and
# subscription viewers group. The groups have already been created.

dn: cn=subscriptions,orclApplicationCommonName=%s_AppName%,cn=%s_
AppType%,cn=Products,
 cn=OracleContext,%s_IdentityRealm%
changetype: modify
replace: orclaci
orclaci: access to entry by group="cn=Subscription_
Admins,cn=Subscriptions,orclApplicationCommonName=%s_AppName%,
 cn=%s_AppType%,cn=products,cn=OracleContext,%s_IdentityRealm%"
(browse,add,delete) by  group="cn=Subscription_
Viewers,cn=Subscriptions,orclApplicationCommonName=%s_AppName%,
 cn=%s_AppType%,cn=products,cn=OracleContext,%s_IdentityRealm%" (browse)
orclaci: access to attr=(*) by group="cn=Subscription_
Admins,cn=Subscriptions,orclApplicationCommonName=%s_AppName%,
 cn=%s_AppType%,cn=products,cn=OracleContext,%s_IdentityRealm%"
(search,read,write,compare) by  group="cn=Subscription_
Viewers,cn=Subscriptions,orclApplicationCommonName=%s_AppName%,
 cn=%s_AppType%,cn=products,cn=OracleContext,%s_IdentityRealm%"
(search,read,compare)
```

Determining Scheduling Parameters for the Employee Self Service Application

The scheduling interval determines how often the provisioning servers send or receive events. The server sends or receives events, and, when it has finished sending or receiving all of them, it sleeps for a period specified in seconds in the scheduling interval. The number of events it can send or receive at one time is dictated by the "Maximum Events per Schedule" parameter.

Let us assume that we need events to be propagated every two minutes, and a maximum of 100 events each time.

Use the following to determine the interface connection information:

- Interface Type: This is the event propagation medium. Currently, only PL/SQL is supported.
- Interface Name: This is the name of the PL/SQL package that the application must implement and that the provisioning server invokes to send and receive events. For our sample application. let us assume `ESSA_INTF` to be the interface name.
- Interface Connection information: This is used by the server to connect to the application database to invoke the PL/SQL interface.

The connection information is in this format:

```
Database Host: Listener Port: Database SID: DB Account: Password
```

For a high-availability, RAC-enabled database, the connection information should be in this format:

```
Database Host: Listener Port: Service Name: DB Account: Password; Database Host:
Listener Port: Service Name: DB Account: Password; Database Host: Listener Port:
Service Name: DB Account: Password
```

The entire string should be specified in one line as a single value.

For our sample application, the connection information is:

```
localhost: 1521: iasdb : scott : tiger
```

The Oracle directory integration and provisioning server uses JDBC to connect to the application database using the connect information provided, and then invokes the PL/SQL APIs to propagate or receive events.

Determining the Interface Connection Information for the Employee Self Service Application

Use the following to determine the interface connection information:

- Interface Type: This is the event propagation medium. Currently only PL/SQL is supported.
- Interface Name: This is the name of the PL/SQL package that the application must implement and that the provisioning server invokes to send and receive events. For our sample application, let us assume that ESSA_INTF is the interface name.
- Interface Connection information: This is used by the server to connect to the application database to invoke the PL/SQL interface.

The following types of Database connection formats are supported:

- Database Host: Listener Port: Database SID: DB Account: Password. This is the old format, which is still supported. Nevertheless, do not use it because SID support might soon be obsolete. In this case, the provisioning server uses JDBC thin driver to connect. For example:

  ```
  localhost: 1521: iasdb: scott: tiger
  ```

- Database Host: Listener Port: DB Service Name: DB Account: Password. This is recommended. In this case, the provisioning server uses JDBC thin driver to connect. For example:

```
localhost: 1521: iasdb: scott: tiger
```

- For a database configured for active failover clusters, the connection information should be in one of the following two formats:

  ```
  DBSVC=netServiceName:User:Password
  ```

  In this case, the provisioning server uses JDBC OCI (thick) driver to connect. The net service name needs to be defined in the local tnsnames.ora file on which the provisioning server is running.

  ```
  DBURL=ldap://ldap_host:ldap_port/DBServiceName:UserName:Password
  ```

  In this case, the provisioning server uses JDBC thin driver to connect. Database Registration should already have occurred in the directory. the driver connects to the directory using the directory host and port. It retrieves the database connect information and then connects to the database.

Implementing the Interface Specification for the Employee Self Service Application

The interface is described in detail in Chapter 13, "Provisioning Integration API Reference".

For outbound events—that is, events from Oracle Internet Directory to the application—the following interfaces must be implemented:

```
PROCEDURE PutOIDEvent (event         IN  LDAP_EVENT,
                       event_status  OUT LDAP_EVENT_STATUS);
```

For inbound events—that is, events from application to Oracle Internet Directory—the following interfaces must be implemented:

```
FUNCTION GetAppEvent(event OUT LDAP_EVENT) RETURNING NUMBER;
PROCEDURE PutAppEventStatus(event_status IN LDAP_EVENT_STATUS)
```

For our sample application, because we are handling only outbound events, we implement all interfaces concerning those events.

Creating the Provisioning Subscription Profile for the Employee Self Service Application

To create the provisioning subscription profile, use the following settings:

```
ORACLE_HOME/bin/oidprovtool operation=create ldap_host=localhost \
ldap_port=389 ldap_user_dn=cn=orcladmin ldap_user_password=welcome \
organization_dn="o=ACME,dc=com" \
application_dn="orclApplicationCommonName=ESSA,cn=demoApps,cn=Products,\
cn=OracleContext,o=ACME,dc=com" \
interface_name=ESSA_INTF  interface_type=PLSQL \
interface_connect_info="localhost:1521:iasdb:scott:tiger" \
event_subscription="IDENTITY:o=oracle,dc=com:ADD(cn,sn,mail,description,
telephonenumber)" \
```

```
event_subscription="IDENTITY:o=oracle,dc=com:MODIFY(cn,sn,mail,description,
telephonenumber)" \
event_subscription="IDENTITY:o=oracle,dc=com:DELETE" \
event_subscription="SUBSCRIPTION:cn=ESSA,cn=products,cn=oraclecontext,o=oracle,
dc=com:ADD(orclactivestartdate,orclactiveenddate,cn) \
event_subscription="SUBSCRIPTION:cn=ESSA,cn=prducts,cn=oraclecontext,o=oracle,
dc=com:MODIFY(orclactivestartdate,orclactiveenddate,cn) \
event_subscription="SUBSCRIPTION:cn=ESSA,cn=prducts,cn=oraclecontext,o=oracle,
dc=com:DELETE"
```

Provisioning Integration Prerequisites

To use the Oracle Directory Provisioning Integration Service, an application must be Oracle RDBMS-based, and it must be enabled for Oracle Application Server Single Sign-On.

As an application developer, you should be familiar with:

- General LDAP concepts
- Oracle Internet Directory
- Oracle Internet Directory integration with Oracle Application Server
- Oracle Delegated Administration Services
- The user provisioning model described in the chapter on the Oracle Directory Provisioning Integration Service in *Oracle Identity Management Integration Guide*.
- The Oracle Directory Integration and Provisioning platform
- Knowledge of SQL, PL/SQL, and database RPCs

In addition, Oracle Corporation recommends that you understand single sign-on concepts.

Development Usage Model for Provisioning Integration

This section shows how an application interacts with the Oracle Directory Provisioning Integration Service. It contains these topics:

- Initiating Provisioning Integration
- Returning Provisioning Information to the Directory

Initiating Provisioning Integration

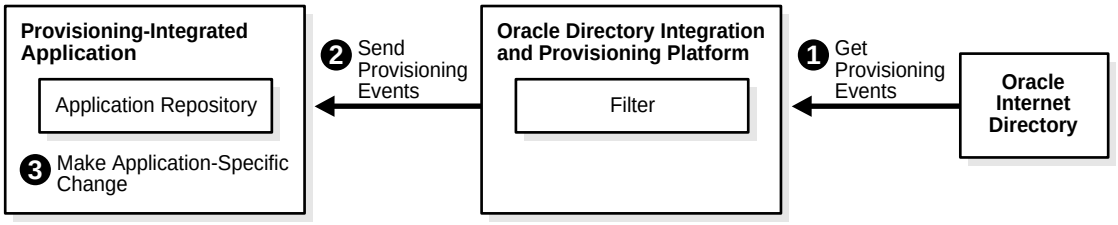
When an application is installed, the Oracle Directory Provisioning Integration Service is provided with three kinds of information:

- Information that enables an application entry to be registered in the directory
- database connect information, which is also registered in the directory
- Information that enables the Oracle Directory Provisioning Integration Service to service the application

Database connect information for the application is also registered.

Figure 4–1 on page 4-13 shows the first phase of provisioning—namely, passing user events from Oracle Internet Directory through the Oracle Directory Integration and Provisioning platform provisioning filter to the application.

Figure 4–1 How an Application Obtains Provisioning Information by Using the Oracle Directory Provisioning Integration Service



In Figure 4–1:

1. The Oracle Directory Provisioning Integration Service retrieves the changes to users and groups from the Oracle Internet Directory change log. It determines which changes to send to the application.
2. The Oracle Directory Provisioning Integration Service sends the changes to the application—using database connect information—by invoking a generic provisioning interface.
3. The generic provisioning interface invokes application-specific logic. The application-specific logic translates the generic provisioning event to one that is application-specific. It then makes the necessary changes in the application repository.

Returning Provisioning Information to the Directory

It is now possible to return provisioning information to Oracle Internet Directory. Figure 4–2 shows the steps involved in this process, which is essentially the reverse of the provisioning process.

1. The application repository generates the application event data and sends it to the Oracle Directory Integration and Provisioning platform.
2. The Oracle Directory Integration and Provisioning platform filters the event data and returns the change information to the directory server.
3. The change is applied in Oracle Internet Directory.

The updated information is stored in Oracle Internet Directory, ready to be accessed by other applications.

Figure 4–2 How an Application Returns Provisioning Information to Oracle Internet Directory Provisioning Service

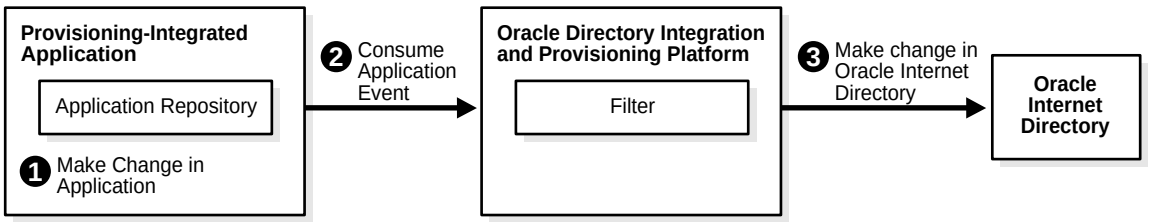


Figure 4–3 on page 4-14 shows the relationship between the services and the subscribed applications in a provisioning-integrated deployment.

Figure 4–3 Provisioning Services and Their Subscribed Applications in a Typical Deployment

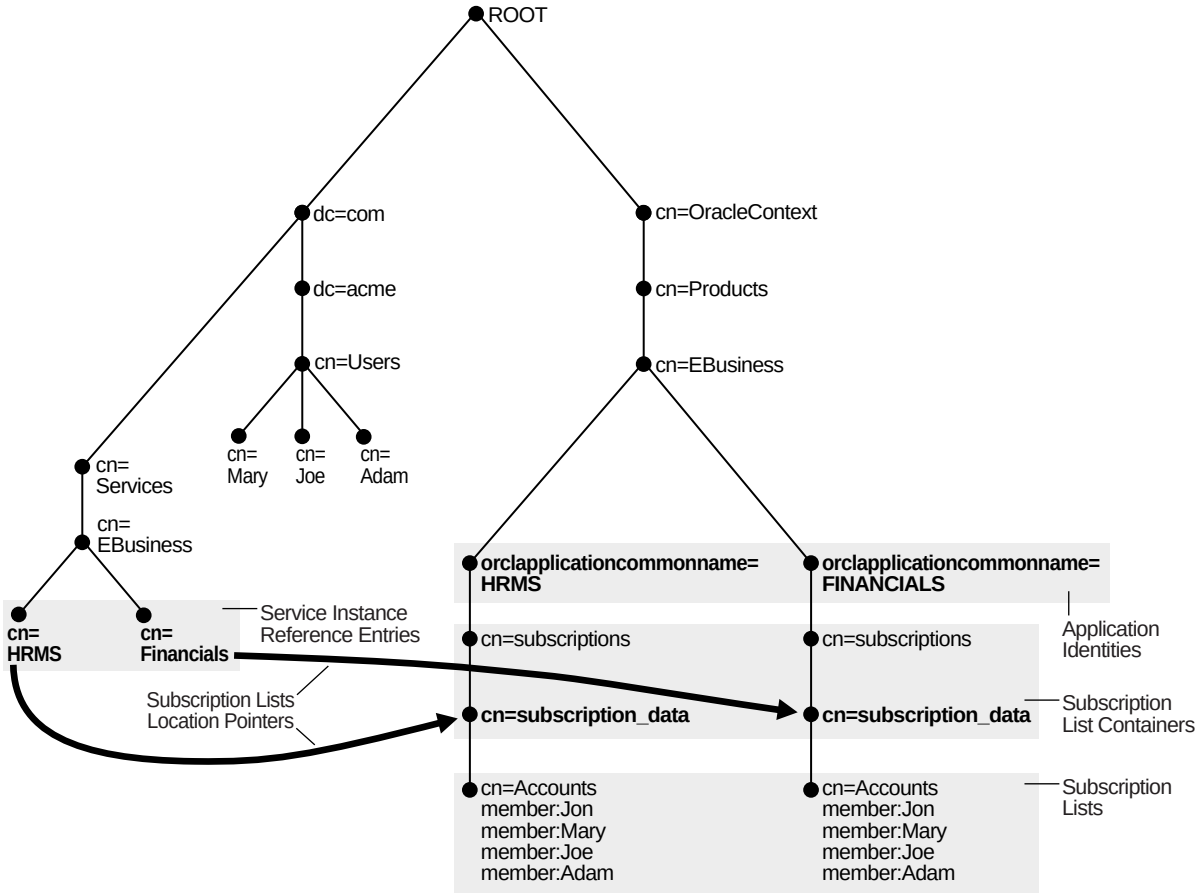


Figure 4–3 shows a DIT in which the entries for two services—Oracle Human Resources and Oracle Financials—point to their corresponding subscription list containers.

- Oracle Human Resources is represented as `cn=HRMS,cn=EBusiness,cn=Services,dc=com`.

  It points to its subscription list: `cn=Accounts,cn=subscription_data, cn=subscriptions,orclapplicationcommonname=HRMS, cn=EBusiness,cn=Products,cn=OracleContext`.

- Oracle Financials is represented as `cn=Financials,cn=EBusiness,cn=Services,dc=com`.

  It points to its subscription list: `cn=Accounts,cn=subscription_ data,cn=subscriptions,orclapplicationcommonname=FINANCIALS, cn=EBusiness,cn=Products,cn=OracleContext`.

Development Tasks for Provisioning Integration

To develop applications for synchronized provisioning, you perform these general tasks:

1. Develop application-specific logic to perform provisioning activities in response to events from the provisioning system.

2. Modify application installation procedures to enable the applications to subscribe to provisioning events.

This section contains these topics:

- Application Installation
- User Creation and Enrollment
- User Deletion
- Extensible Event Definitions
- Application Deinstallation

Application Installation

Modify the installation logic for each application to run a post-installation configuration tool.

During application installation, the application invokes the Provisioning Subscription Tool (`oidprovtool`). The general pattern of invoking this tool is:

```
oidprovtool param1=p1_value param2=p2_value param3=p3_value ...
```

> **See Also:** "Development Usage Model for Provisioning Integration" on page 4-12 for details of what the post-installation tool should do

User Creation and Enrollment

First, create users in Oracle Internet Directory; then enroll them in the application.

When using either of these interfaces, you must enable the Oracle Directory Provisioning Integration Service to identify users presently enrolled in the application. This way, the delete events it sends correspond only to users enrolled in the application.

Implement the application logic so that the `user_exists` function verifies that a given user in Oracle Internet Directory is enrolled in the application.

User Deletion

The Oracle Directory Provisioning Integration Service primarily propagates the user deletion events from Oracle Internet Directory to the various provisioning-integrated applications.

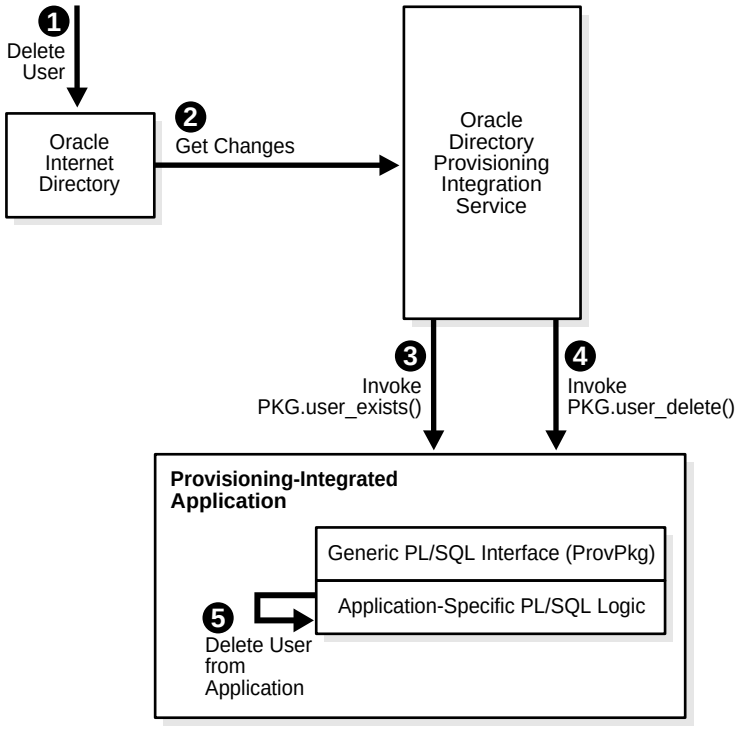
Using the PL/SQL callback interface, the application registers with the Oracle Directory Provisioning Integration Service and provides:

- The name of a PL/SQL package the application is using
- The connect string to access that package

The Oracle Directory Provisioning Integration Service in turn connects to the application database and invokes the necessary PL/SQL procedures.

Figure 4–4 on page 4-16 illustrates system interactions for the PL/SQL callback interface.

Figure 4–4 PL/SQL Callback Interface



As Figure 4–4 shows, deleting a user from an application comprises these steps:

1. The administrator deletes the user in Oracle Internet Directory by using Oracle Directory Manager or a similar tool.
2. The Oracle Directory Provisioning Integration Service retrieves that change from the Oracle Internet Directory change-log interface.
3. To see if the user deleted from the directory was enrolled for this application, the Oracle Directory Provisioning Integration Service invokes the `user_exists()` function of the provisioning event interface of the application.
4. If the user is enrolled, the Oracle Directory Provisioning Integration Service invokes the `user_delete()` function of the provisioning event interface.
5. The application-specific PL/SQL logic deletes the user and the related footprint from the application-specific repository. This step is the responsibility of the developer.

Extensible Event Definitions

This feature enables you to extend the abilities of the Oracle Directory Provisioning Integration Service to return predefined sets of provisioning information to applications. Configure the following events at installation to propagate them to the appropriate applications.

Table 4–1 Extensible Event Definitions

Event Definition	Attribute
Event Object Type (`orclODIPProvEventObjectType`)	Specifies the type of object the event is associated with—for example, `USER`, `GROUP`, or `IDENTITY`.
LDAP Change Type (`orclODIPProvEventChangeType`)	Indicates what kinds of LDAP operations can generate an event for this type of object—for example, `ADD`, `MODIFY`, or `DELETE`)
Event Criteria (`orclODIPProvEventCriteria`)	The additional selection criteria that qualifies an LDAP entry to be of a specific object type. For example, `Objectclass=orclUserV2` means that any LDAP entry that satisfies this criteria can be qualified as this object type, and any change to this entry can generate appropriate event(s).

Application Deinstallation

You must enable the deinstallation logic for each provisioning-integrated application to run the Provisioning Subscription Tool (`oidprovtool`) that unsubscribes the application from the Oracle Directory Provisioning Integration Service.

LDAP_NTFY Function Definitions

The Oracle Directory Provisioning Integration Service invokes the following callback functions.

FUNCTION user_exists

A callback function invoked by the Oracle Directory Provisioning Integration Service to check if a user is enrolled with the application.

Syntax

```
FUNCTION user_exists ( user_name    IN VARCHAR2,
 user_guid    IN VARCHAR2,
 user_dn      IN VARCHAR2)
```

Parameters

Table 4–2 Function user_exists Parameters

Parameter	Description
`user_name`	User identifier
`user_guid`	Global user identifier
`user_dn`	DN attribute of the user entry

Return Value

Returns a positive number if the user exists

FUNCTION group_exists

A callback function invoked by the Oracle Directory Provisioning Integration Service to check whether a group exists in the application.

Syntax

```
FUNCTION group_exists ( group_name IN VARCHAR2,
```

```
group_guid IN VARCHAR2,
group_dn   IN VARCHAR2)
RETURN NUMBER;
```

Parameters

Table 4–3 Function group_exists Parameters

Parameter	Description
group_name	Group simple name
group_guid	GUID of the group
group_dn	DN of the group entry

Return value

Returns a positive number if the group exists. Returns zero if the group does not exist.

FUNCTION event_ntfy

A callback function invoked by the Oracle Directory Provisioning Integration Service to deliver change notification events for objects modeled in Oracle Internet Directory. Currently modify and delete change notification events are delivered for users and groups in Oracle Internet Directory. While delivering events for an object (represented in Oracle Internet Directory), the related attributes are also sent along with other details. The attributes are delivered as a collection (array) of attribute containers, which are in un-normalized form. This means that, if an attribute has two values, two rows are sent in the collection.

Syntax

```
FUNCTION event_ntfy ( event_type  IN VARCHAR2,
event_id    IN VARCHAR2,
event_src   IN VARCHAR2,
event_time  IN VARCHAR2,
object_name IN VARCHAR2,
object_guid IN VARCHAR2,
object_dn   IN VARCHAR2,
profile_id  IN VARCHAR2,
attr_list   IN LDAP_ATTR_LIST )
RETURN NUMBER;
```

Parameters

Table 4–4 Parameters for FUNCTION event_ntfy

Parameter	Description
event_type	Type of event. Possible values: USER_DELETE, USER_MODIFY, GROUP_DELETE, GROUP_MODIFY.
event_id	Event id (change log number).
event_src	DN of the modifier responsible for this event.
event_time	Time when this event occurred.
object_name	Simple name of the entry.
object_guid	GUID of the entry.
object_dn	DN of the entry

Table 4–4 (Cont.) Parameters for FUNCTION event_ntfy

Parameter	Description
`profile_id`	Name of the Provisioning Agent
`attr_list`	Collection of ldap attributes of the entry

Return Values

Success returns a positive number. Failure returns zero.

5

Developing Directory Plug-ins

This chapter explains how to use the plug-in framework for Oracle Internet Directory to extend LDAP operations.

This chapter contains these topics:

Plug-in Prerequisites

To develop Oracle Internet Directory plug-ins, you should be familiar with the following:

- Generic LDAP concepts
- Oracle Internet Directory
- Oracle Internet Directory integration with Oracle Application Server
- SQL, PL/SQL, and database RPCs

Plug-in Benefits

To extend the capabilities of the Oracle Internet Directory server, you can write your own server plug-in. A server plug-in is a PL/SQL package, shared object or library, or a dynamic link library on Windows NT that contains your own functions. Oracle supports only PL/SQL plug-ins.

You can extend LDAP operations in the following ways:

- You can validate data before the server performs an LDAP operation on the data
- You can perform actions (that you define) after the server successfully completes an LDAP operation
- You can define extended operations

- You can be authenticated through external credential stores
- You can replace an existing server module by defining your own server module

For the last one, you may, for example, implement your own password value checking and place it into the Oracle Internet Directory server.

On startup, the directory server loads your plug-in configuration and library. It calls your plug-in functions while processing various LDAP requests.

> **See Also:** The chapter about the password policy plug-in in *Oracle Internet Directory Administrator's Guide*. The chapter contains an example of how to implement your own password value checking and place it into the Oracle Internet Directory server.

What Is the Plug-in Framework?

The plug-in framework is the environment in which the plug-in user can develop, configure, and apply the plug-ins. Each individual plug-in instance is called a plug-in module.

The plug-in framework includes the following:

- Plug-in configuration tools
- Plug-in module interface
- Plug-in LDAP API (`ODS.LDAP_PLUGIN` package)

Follow these steps to use the server plug-in framework:

1. Write a user-defined plug-in procedure. This plug-in module must be written in PL/SQL.
2. Compile the plug-in module against the same database that serves as the Oracle Internet Directory backend database.
3. Grant execute permission of the plug-in module to ods_server.
4. Register the plug-in module through the configuration entry interface.

Operation-Based Plug-ins Supported by the Directory

For operation-based plug-ins, there are pre-operation, post-operation, and when-operation plug-ins.

Pre-Operation Plug-ins

The server calls pre-operation plug-in modules before performing the LDAP operation. The main purpose of this type of plug-in is to validate data before the data can be used in the LDAP operation.

When an exception occurs in the pre-operation plug-in, one of the following occurs:

- When the return error code indicates warning status, the associated LDAP request proceeds.
- When the return code indicates failure status, the request does not proceed.

If the associated LDAP request fails later on, the directory does not roll back the committed code in the plug-in modules.

Post-Operation Plug-ins

The Oracle Internet Directory server calls post-operation plug-in modules after performing an LDAP operation. The main purpose of this type of plug-in is to invoke a function after a particular LDAP operation is executed. For example, logging and notification are post-operation plug-in functions.

When an exception occurs in the post-operation plug-in, the associated LDAP operation is not rolled back.

If the associated LDAP request fails, the post plug-in is still executed.

When-Operation Plug-ins

The directory calls when-operation plug-in modules while performing standard LDAP operations. The main purpose of this type of plug-in is to augment existing operations within the same LDAP transaction. If either the LDAP request or the plug-in program fails, all the changes are rolled back.

There are different types of When-operation plug-ins.

- Add-on
- Replace

You can, for example, use both add-on and replace plug-ins with the ldapcompare operation. If you use the first type, the directory executes its server compare code and executes the plug-in module defined by the plug-in developer. If you use the second type, the directory does not execute its compare code. Instead it relies on the plug-in module to perform the comparison.

Replace plug-ins are supported only in `ldapadd`, `ldapcompare`, `ldapdelete`, `ldapmodify`, and `ldapbind`. Add-on plug-ins are supported in `ldapadd`, `ldapdelete`, and `ldapmodify`.

Designing, Creating, and Using Plug-ins

This section contains these topics:

Designing Plug-ins

Use the following guidelines when designing plug-ins:

- Use plug-ins to guarantee that when a specific LDAP operation is performed, related actions are also performed.
- Use plug-ins only for centralized, global operations that should be invoked for the program body statement, regardless of which user or LDAP application issues the statement.
- Do not create recursive plug-ins. For example, creating a `PRE_LDAP_BIND` plug-in that itself issues an `ldapbind` (through the `DBMS_LDAP` PL/SQL API) statement, causes the plug-in to execute recursively until it has run out of resources.

> **Note:** Use plug-ins on the LDAP PL/SQL API judiciously. They are executed for every LDAP request every time the event occurs on which the plug-in is created.

Types of Plug-in Operations

A plug-in can be associated with `ldapbind`, `ldapadd`, `ldapmodify`, `ldapcompare`, `ldapsearch`, and `ldapdelete` operations.

Naming Plug-ins

Plug-in names (PL/SQL package names) must be unique if they share the same database schema with other plug-ins or stored procedures. But plug-ins can share names with other database schema objects such as tables and views. This kind of sharing is not, however, recommended.

Creating Plug-ins

Creating a plug-in module is like creating a PL/SQL package. Both have a specification part and a body part. The directory, not the plug-in, defines the plug-in specification because the specification serves as the interface between Oracle Internet Directory and the custom plug-in.

For security reasons and for the integrity of the LDAP server, you can compile plug-ins only in the ODS database schema. You must compile them in the database that serves as the backend database of Oracle Internet Directory.

Package Specifications for Plug-in Module Interfaces

Different plug-ins have different package specifications. As Table 5–1 shows, you can name the plug-in package. You must, however, follow the signatures defined for each type of plug-in procedure. See "Specifications for Plug-in Procedures" for details.

Table 5–1 Plug-in Module Interface

Plug-in Item	User Defined	Oracle Internet Directory-Defined
Plug-in Package Name	X	
Plug-in Procedure Name		X
Plug-in Procedure Signature		X

Table 5–2 names the different plug-in procedures. In addition, it lists and describes the parameters that these procedures use.

Table 5–2 Operation-Based and Attribute-Based Plug-in Procedure Signatures

Invocation Context	Procedure Name	IN Parameters	OUT Parameters
Before `ldapbind`	`PRE_BIND`	ldapcontext, Bind DN, Password	return code, error message
With `ldapbind` but replacing the default server behavior	`WHEN_BIND_REPLACE`	ldapcontext, bind result, DN, userpassword	bind result, return code, error message
After `ldapbind`	`POST_BIND`	ldapcontext, Bind result, Bind DN, Password	return code, error message
Before `ldapmodify`	`PRE_MODIFY`	ldapcontext, DN, Mod structure	return code, error message
With `ldapmodify`	`WHEN_MODIFY`	ldapcontext, DN, Mod structure	return code, error message
With `ldapmodify` but replacing the default server behavior	`WHEN_MODIFY_REPLACE`	ldapcontext, DN, Mod structure	return code, error message
After `ldapmodify`	`POST_MODIFY`	ldapcontext, Modify result, DN, Mod structure	return code, error message
Before `ldapcompare`	`PRE_COMPARE`	ldapcontext, DN, attribute, value	return code, error message
With `ldapcompare` but replacing the default server behavior	`WHEN_COMPARE_ REPLACE`	ldapcontext, Compare result, DN, attribute, value	compare result, return code, error message
After `ldapcompare`	`POST_COMPARE`	ldapcontext, Compare result, DN, attribute, value	return code, error message
Before `ldapadd`	`PRE_ADD`	ldapcontext, Entry	return code, error message
With `ldapadd`	`WHEN_ADD`	ldapcontext, Entry	return code, error message
With `ldapadd` but replacing the default server behavior	`WHEN_ADD_REPLACE`	ldapcontext, Entry	return code, error message
After `ldapadd`	`POST_ADD`	ldapcontext, Add result, Entry	return code, error message
Before `ldapdelete`	`PRE_DELETE`	ldapcontext, DN	return code, error message
With `ldapdelete`	`WHEN_DELETE`	ldapcontext, DN	return code, error message
With `ldapdelete` but replacing the default server behavior	`WHEN_DELETE`	ldapcontext, DN	return code, error message
After `ldapdelete`	`POST_DELETE`	ldapcontext, Delete result, DN	return code, error message
Before `ldapsearch`	`PRE_SEARCH`	ldapcontext, Base DN, scope, filter	return code, error message

Table 5–2 (Cont.) Operation-Based and Attribute-Based Plug-in Procedure Signatures

Invocation Context	Procedure Name	IN Parameters	OUT Parameters
After `ldapsearch`	`POST_SEARCH`	Ldap context, Search result, Base DN, scope, filter	return code, error message

See Also:

- "Error Handling" on page 5-10 for valid values for the return code and error message
- "Specifications for Plug-in Procedures" on page 5-26 for complete supported procedure signatures

Compiling Plug-ins

Plug-ins are exactly the same as PL/SQL stored procedures. A PL/SQL anonymous block is compiled each time it is loaded into memory. Compilation consists of these stages:

1. Syntax checking: PL/SQL syntax is checked, and a parse tree is generated.
2. Semantic checking: Type checking and further processing on the parse tree.
3. Code generation: The pcode is generated.

If errors occur during the compilation of a plug-in, the plug-in is not created. You can use the `SHOW ERRORS` statement in SQL*Plus or Enterprise Manager to see any compilation errors when you create a plug-in, or you can `SELECT` the errors from the `USER_ERRORS` view.

All plug-in modules must be compiled in the ODS database schema.

Dependencies

Compiled plug-ins have dependencies. They become invalid if an object depended upon, such as a stored procedure or function called from the plug-in body, is modified. Plug-ins that are invalidated for dependency reasons must be recompiled before the next invocation.

Recompiling Plug-ins

Use the `ALTER PACKAGE` statement to manually recompile a plug-in. For example, the following statement recompiles the my_plugin plug-in:

```
ALTER PACKAGE my_plugin COMPILE PACKAGE;
```

Granting Permission

Use the `GRANT EXECUTE` statement to grant execute permission to ods_server for the plug-in modules.

Registering Plug-ins

To enable the directory server to call a plug-in at the right moment, you must register the plug-in with the directory server. Do this by creating an entry for the plug-in under `cn=plugin,cn=subconfigsubentry`.

The orclPluginConfig Object Class

A plug-in must have `orclPluginConfig` as one of its object classes. This is a structural object class, and its super class is `top`. Table 5–3 lists and describes its attributes.

Table 5–3 Plug-in Attribute Names and Values

Attribute Name	Attribute Value	Mandatory?
`cn`	Plug-in entry name	Yes
`orclPluginAttributeList` (only for `ldapcompare` and `ldapmodify` plug-ins.)	A semicolon-separated list of attribute names that controls whether the plug-in takes effect. If the target attribute is included in the list, then the plug-in is invoked.	No
`orclPluginEnable`	`0` = disable (default) `1` = enable	No
`orclPluginEntryProperties`	An `ldapsearch` filter type value must be specified. For example, if we specify `orclPluginEntryProperties: (&(objectclass=inetorgperson)(sn=Cezanne))`, the plug-in is not invoked if the target entry has `objectclass` equal to `inetorgperson` and `sn` equal to `Cezanne`.	No
`orclPluginIsReplace`	`0` = disable (default) `1` = enable For `WHEN` timing plug-in only	No
`orclPluginKind`	PL/SQL	No

`orclPluginLDAPOperation`	One of the following values:
	`ldapcompare`
	`ldapmodify`
	`ldapbind`
	`ldapadd`
	`ldapdelete`
	`ldapsearch`
		Yes

`orclPluginName`	Plug-in package name	Yes
`orclPluginRequestGroup`	A semicolon-separated group list that controls if the plug-in takes effect. You can use this group to specify who can actually invoke the plug-in. For example, if you specify `orclpluginrequestgroup:cn=security,cn=groups,dc=oracle,dc=com` when you register the plug-in, the plug-in will not be invoked unless the ldap request comes from the person who belongs to the group `cn=security,cn=groups,dc=oracle,dc=com`.	No

Table 5–3 (Cont.) Plug-in Attribute Names and Values

Attribute Name	Attribute Value	Mandatory?
`orclPluginRequestNegGroup`	A semicolon-separated group list that controls if the plug-in takes effect. You can use this group to specify who cannot invoke the plug-in. For example, if you specify `orclpluginrequestgroup: cn=security,cn=groups,dc=oracle,dc=com`, when you register the plug-in, the plug-in is not invoked if the LDAP request comes from the person who belongs to the group `cn=security,cn=groups,dc=oracle,dc=com`.	No
`orclPluginResultCode`	An integer value to specify the ldap result code. If this value is specified, then plug-in will be invoked only if the LDAP operation is in that result code scenario. This is only for the post plug-in type.	No
`orclPluginShareLibLocation`	File location of the dynamic linking library. If this value is not present, then Oracle Internet Directory server assumes the plug-in language is PL/SQL.	No
`orclPluginSubscriberDNList`	A semicolon separated DN list that controls if the plug-in takes effect. If the target DN of an LDAP operation is included in the list, then the plug-in is invoked.	No

`orclPluginTiming`	One of the following values:
	`pre`
	`when`
	`post`
		No

`orclPluginType`	One of the following values:
	`operational`
	`attribute`
	`password_policy`
	`syntax`
	`matchingrule`
	See Also: Operation-Based Plug-ins Supported by the Directory on page 5-2
		Yes

`orclPluginVersion`	Supported plug-in version number	No

Adding a Plug-in Configuration Entry by Using Command-Line Tools

Plug-ins must be added to Oracle Internet Directory server so that the server is aware of additional operations that must be performed at the correct time.

When the plug-in successfully compiles against the Oracle Internet Directory backend database, create a new entry and place it under `cn=plugin,cn=subconfigsubentry`.

In the following examples, an entry is created for an operation-based plug-in called `my_plugin1`. The LDIF file, `my_ldif_file.ldif`, is as follows:

Example 1

The following is an example LDIF file to create such an object:

```
cn=when_comp,cn=plugin,cn=subconfigsubentry
```

```
objectclass=orclPluginConfig
objectclass=top
orclPluginName=my_plugin1
orclPluginType=operational
orclPluginTiming=when
orclPluginLDAPOperation=ldapcompare
orclPluginEnable=1
orclPluginVersion=1.0.1
orclPluginIsReplace=1
cn=when_comp
orclPluginKind=PLSQL
orclPluginSubscriberDNList=dc=COM,c=us;dc=us,dc=oracle,dc=com;dc=org,dc=us;o=IMC
,c=US
orclPluginAttributeList=userpassword
```

Example 2

```
cn=post_mod_plugin, cn=plugin,cn=subconfigsubentry
objectclass=orclPluginConfig
objectclass=top
orclPluginName=my_plugin1
orclPluginType=operational
orclPluginTiming=post
orclPluginLDAPOperation=ldapmodify
orclPluginEnable=1
orclPluginVersion=1.0.1
cn=post_mod_plugin
orclPluginKind=PLSQL
```

Add this file to the directory with the following command:

```
ldapadd -p 389 -h myhost -D binddn -w password -f my_ldif_file.ldif
```

Note: To avoid creating an inconsistent state, the plug-in configuration entry is not replicated.

Managing Plug-ins

This section explains how to modify and debug plug-ins.

Modifying Plug-ins

Like a stored procedure, a plug-in cannot be explicitly altered. It must be replaced with a new definition.

When replacing a plug-in, you must include the `OR REPLACE` option in the `CREATE PACKAGE` statement. The `OR REPLACE` option enables a new version of an existing plug-in to replace an older version without having an effect on grants made for the original version of the plug-in.

Alternatively, the plug-in can be dropped using the `DROP PACKAGE` statement, and you can rerun the `CREATE PACKAGE` statement.

If the plug-in name (the package name) is changed, you must register the new plug-in again.

Debugging Plug-ins

You can debug a plug-in using the same facilities available for PL/SQL stored procedures.

Enabling and Disabling Plug-ins

To turn the plug-in on or off, modify the value of `orclPluginEnable` in the plug-in configuration object. For example, modify the value of `orclPluginEnable` in `cn=post_mod_plugin,cn=plugins,cn=subconfigsubentry` to be `1` or `0`.

Exception Handling

Each of the procedures in a PL/SQL plug-in must have an exception handling block that handles errors intelligently and, if possible, recovers from them.

Error Handling

Oracle Internet Directory requires that the return code (`rc`) and error message (`errmsg`) be set correctly in the plug-in procedures.

Table 5–4 provides the values that are valid for the return code.

Table 5–4 Valid Values for the Plugin-in Return Code

Error Code	Description
`0`	Success
Any number greater than zero	Failure
`-1`	Warning

The `errmsg` parameter is a string value that can pass a user's custom error message back to Oracle Internet Directory server. The size limit for `errmsg` is 1024 bytes. Each time Oracle Internet Directory runs the plug-in program, it examines the return code to determine if it must display the error message.

If, for example, the value for the return code is `0`, the error message value is ignored. If the value of the return code is `-1` or greater than zero, the following message is either logged in the log file or displayed in standard output if the request came from LDAP command-line tools:

```
ldap addition info: customized error
```

Program Control Handling between Oracle Internet Directory and Plug-ins

Table 5–5 shows where plug-in exceptions occur and how the directory handles them.

Table 5–5 Program Control Handling when a Plug-in Exception Occurs

Plug-in Exception Occurred in	Oracle Internet Directory Server Handling
`PRE_BIND`, `PRE_MODIFY`, `PRE_ADD`, `PRE_SEARCH`, `PRE_COMPARE`, `PRE_DELETE`	Depends on return code. If the return code is: ▪ Greater than zero (error), then no LDAP operation is performed ▪ `-1` (warning), then proceed with the LDAP operation

Table 5–5 (Cont.) Program Control Handling when a Plug-in Exception Occurs

Plug-in Exception Occurred in	Oracle Internet Directory Server Handling
POST_BIND, POST_MODIFY, POST_ADD, POST_SEARCH, WHEN_DELETE	LDAP operation is completed. There is no rollback.
WHEN_MODIFY, WHEN_ADD, WHEN_DELETE	Rollback the LDAP operation

Table 5–6 shows how the directory responds when an LDAP operation fails.

Table 5–6 Program Control Handling when an LDAP Operation Fails

LDAP Operation Fails in	Oracle Internet Directory Server Handling
PRE_BIND, PRE_MODIFY, PRE_ADD, PRE_SEARCH, WHEN_DELETE	Pre-operation plug-in is completed. There is no rollback.
POST_BIND, POST_MODIFY, POST_ADD, POST_SEARCH, WHEN_DELETE	Proceed with post-operation plug-in. The LDAP operation result is one of the IN parameters.
WHEN_MODIFY, WHEN_ADD, WHEN_DELETE	When types of plug-in changes are rolled back.
WHEN	Changes made in the plug-in program body are rolled back.

Plug-in LDAP API

There are different methods for providing API access:

- Enable a user to utilize the standard LDAP PL/SQL APIs. Note though that, if program logic is not carefully planned, an infinite loop in plug-in execution can result.
- Oracle Internet Directory provides the Plug-in LDAP API. This plug-in does not cause a series of plug-in actions in the directory server if there are plug-ins configured and associated with the LDAP request.

In the Plug-in LDAP API, the directory provides APIs for connecting back to the directory server designated in the plug-in module. You must use this API if you want to connect to the server that is executing the plug-in. If you want to connect to an external server, you can use the DBMS_LDAP API.

Within each plug-in module, an `ldapcontext` is passed from the Oracle directory server. When the Plug-in LDAP API is called, `ldapcontext` is passed for security and binding purposes. When binding with this `ldapcontext`, Oracle Internet Directory recognizes that the LDAP request is coming from a plug-in module. For this type of plug-in bind, the directory does not trigger any subsequent plug-ins. It handles the plug-in bind as a super-user bind. Use this plug-in bind with discretion.

See Also: "Plug-in LDAP API Specifications" on page 5-13

Plug-ins and Replication

These cases can cause an inconsistent state in a replication environment:

- Plug-in metadata replicated to other nodes

- Changes to directory entries by plug-in programs or other LDAP operations
- Plug-in installation on only some of the participating nodes
- Implementation in the plug-in of extra checking that depends on the directory data

Plug-in and Database Tools

Bulk tools do not support server plug-ins.

Security

Some Oracle Internet Directory server plug-ins require that you supply the code that preserves tight security. For example, if you replace the directory's `ldapcompare` or `ldapbind` operation with your own plug-in module, you must ensure that your implementation of this operation does not omit any functionality on which security relies.

To ensure tight security, the following must be done:

- Create the plug-in packages
- Only the LDAP administrator can restrict the database user
- Use the access control list (ACL) to set the plug-in configuration entries to be accessed only by the LDAP administrator
- Be aware of the program relationship between different plug-ins

Plug-in Debugging

Use the plug-in debugging mechanism for Oracle Internet Directory to examine the process and content of plug-ins.The following commands control the operation of the server debugging process.

- To set up plug-in debugging, run this command:

  ```
  % sqlplus ods/password @$ORACLE/ldap/admin/oidspdsu.pls
  ```

- To enable plug-in debugging, run this command:

  ```
  % sqlplus ods/password @$ORACLE/ldap/admin/oidspdon.pls
  ```

- After enabling plug-in debugging, you can use this command in the plug-in module code:

  ```
  plg_debug('debuggingmessage');
  ```

  The resulting debug message is stored in the plug-in debugging table.

- To disable debugging, run this command:

  ```
  % sqlplus ods/password @$ORACLE/ldap/admin/oidspdof.pls
  ```

- To display the debug messages that you put in the plug-in module, run this command:

  ```
  % sqlplus ods/password @$ORACLE/ldap/admin/oidspdsh.pls
  ```

- To delete all of the debug messages from the debug table, run this command:

  ```
  % sqlplus ods/password @$ORACLE/ldap/admin/oidspdde.pls
  ```

Plug-in LDAP API Specifications

Here is the package specification that Oracle Internet Directory provides for the Plug-in LDAP API:

```
CREATE OR REPLACE  PACKAGE LDAP_PLUGIN AS
    SUBTYPE SESSION IS RAW(32);

    -- Initializes the LDAP library and return a session handler
    -- for use in subsequent calls.
    FUNCTION init (ldappluginctx IN ODS.plugincontext)
      RETURN SESSION;

    -- Synchronously authenticates to the directory server using
    -- a Distinguished Name and password.
    FUNCTION simple_bind_s (ldappluginctx IN ODS.plugincontext,
                            ld            IN SESSION)
      RETURN PLS_INTEGER;

    -- Get requester info from the plugin context
    FUNCTION get_requester (ldappluginctx IN ODS.plugincontext)
      RETURN VARCHAR2;
END LDAP_PLUGIN;
```

Examples of Plug-ins

This section presents two sample plug-ins. One logs all `ldapsearch` commands. The other synchronizes two directory information trees (DITs).

Example 1: Search Query Logging

Situation: A user wants to know if it is possible to log all of the `ldapsearch` commands.

Solution: Yes. The user can use the post `ldapsearch` operational plug-in for this purpose. They can either log all of the requests or only those that occur under the DNs being searched.

To log all the `ldapsearch` commands:

1. Log all of the `ldapsearch` results into a database table. This log table has these columns:
   - timestamp
   - baseDN
   - search scope
   - search filter
   - required attribute
   - search result

   Use this SQL script to create the table:

   ```
   drop table search_log;
   create table search_log
   (timestamp varchar2(50),
   basedn varchar2(256),
   searchscope number(1);
   searchfilter varchar2(256);
   ```

```
searchresult number(1));
drop table simple_tab;
create table simple_tab (id NUMBER(7), dump varchar2(256));
DROP sequence seq;
CREATE sequence seq START WITH 10000;
commit;
```

2. Create the plug-in package specification.

```
CREATE OR REPLACE PACKAGE LDAP_PLUGIN_EXAMPLE1 AS
PROCEDURE post_search
(ldapplugincontext IN  ODS.plugincontext,
result       IN  INTEGER,
baseDN       IN  VARCHAR2,
scope        IN  INTEGER,
filterStr    IN  VARCHAR2,
requiredAttr IN  ODS.strCollection,
rc           OUT INTEGER,
errormsg     OUT VARCHAR2
);
END LDAP_PLUGIN_EXAMPLE1;
/
```

3. Create the plug-in package body.

```
CREATE OR REPLACE PACKAGE BODY LDAP_PLUGIN_EXAMPLE1 AS
PROCEDURE post_search
(ldapplugincontext IN  ODS.plugincontext,
result       IN  INTEGER,
baseDN       IN  VARCHAR2,
scope        IN  INTEGER,
filterStr    IN  VARCHAR2,
requiredAttr IN  ODS.strCollection,
rc           OUT INTEGER,
errormsg     OUT VARCHAR2
)
      IS
BEGIN
   INSERT INTO simple_tab VALUES
(to_char(sysdate, 'Month DD, YYYY HH24:MI:SS'), baseDN, scope, filterStr,
result);
   -- The following code segment demonstrate how to iterate
   -- the ODS.strCollection
   FOR l_counter1 IN 1..requiredAttr.COUNT LOOP
      INSERT INTO simple_tab
        values (seq.NEXTVAL, 'req attr ' || l_counter1 || ' = ' ||
                requiredAttr(l_counter1));
   END LOOP;
   rc := 0;
   errormsg := 'no post_search plugin error msg';
   COMMIT;
EXCEPTION
   WHEN others THEN
      rc := 1;
      errormsg := 'exception: post_search plguin';
END;
END LDAP_PLUGIN_EXAMPLE1;
/
```

4. Grant permission to `ods_server`.

```
GRANT EXECUTE ON LDAP_PLUGIN_EXAMPLE1 TO ods_server;
```

5. Register the plug-in entry in Oracle Internet Directory.

```
cn=post_search,cn=plugin,cn=subconfigsubentry
objectclass=orclPluginConfig
objectclass=top
orclPluginName=ldap_plugin_example1
orclPluginType=operational
orclPluginTiming=post
orclPluginLDAPOperation=ldapsearch
orclPluginEnable=1
orclPluginVersion=1.0.1
cn=post_search
orclPluginKind=PLSQL
```

Using the `ldapadd` command-line tool to add this entry:

```
% ldapadd -p port_number -h host_name -D bind_dn -w passwd -v -f register_
post_search.ldif
```

Example 2: Synchronizing Two DITs

Situation: There are two interdependent products under `cn=Products`, `cn=oraclecontext`. This interdependency extends down to the users in these products' containers. If a user in the first DIT (product 1) is deleted, the corresponding user in the other DIT (product 2) must be deleted.

Is it possible to set a trigger that, when the user in the first DIT is deleted, calls or passes a trigger to delete the user in the second DIT?

Solution: Yes, we can use the post `ldapdelete` operation plug-in to handle the second deletion occurring in the second DIT.

If the first DIT has the naming context of `cn=DIT1,cn=products,cn=oraclecontext` and the second DIT has the naming context of `cn=DIT2,cn=products,cn=oraclecontext`, the two users share the same ID attribute. Inside of the post `ldapdelete` plug-in module, we can use `LDAP_PLUGIN` and `DBMS_LDAP` APIs to delete the user in the second DIT.

We must set `orclPluginSubscriberDNList` to `cn=DIT1,cn=products,cn=oraclecontext`, so that whenever we delete entries under `cn=DIT1,cn=products,cn=oraclecontext`, the plug-in module is invoked.

Note: When you use a post `ldapmodify` plug-in to synchronize changes between two Oracle Internet Directory nodes, you cannot push all the attributes from one node to the other. This is because the changes (mod structure) captured in the plug-in module include operational attributes. These operational attributes are generated on each node and cannot be modified by using the standard LDAP methods.

When writing your plug-in program, exclude the following operational attributes from synchronization: `authPassword`, `creatorsname`, `createtimestamp`, `modifiersname`, `modifytimestamp`, `pwdchangedtime`, `pwdfailuretime`, `pwdaccountlockedtime`, `pwdexpirationwarned`, `pwdreset`, `pwdhistory`, `pwdgraceusetime`.

The following attributes are used the most in the deployment environment and should be excluded from synchronization first: `pwdchangedtime`, `pwdfailuretime`, `authpassword`, `pwdaccountlockedtime`.

1. Assume that the entries under both DITs have been added to the directory. For example, the entry `id=12345,cn=DIT1,cn=products,cn=oraclecontext` is in `DIT1`, and `id=12345,cn=DIT2,cn=products,cn=oraclecontext` is in `DIT2`.
2. Create the plug-in package specification.

```
CREATE OR REPLACE PACKAGE LDAP_PLUGIN_EXAMPLE2 AS
PROCEDURE post_delete
(ldapplugincontext IN  ODS.plugincontext,
result   IN  INTEGER,
dn       IN  VARCHAR2,
rc       OUT INTEGER,
errormsg OUT VARCHAR2
);
END LDAP_PLUGIN_EXAMPLE2;
/
```

3. Create the plug-in package body.

```
CREATE OR REPLACE PACKAGE BODY LDAP_PLUGIN_EXAMPLE2 AS
PROCEDURE post_delete
(ldapplugincontext IN  ODS.plugincontext,
result   IN  INTEGER,
dn       IN  VARCHAR2,
rc       OUT INTEGER,
errormsg OUT VARCHAR2
)
    IS
     retval       PLS_INTEGER;
     my_session   DBMS_LDAP.session;
     newDN        VARCHAR2(256);
BEGIN
   retval         := -1;
   my_session := LDAP_PLUGIN.init(ldapplugincontext);
   -- bind to the directory
   retval := LDAP_PLUGIN.simple_bind_s(ldapplugincontext, my_session);
   -- if retval is not 0, then raise exception
   newDN := REPLACE(dn, 'DIT1', 'DIT2');
```

```
   retval := DBMS_LDAP.delete_s(my_session, newDN);
   -- if retval is not 0, then raise exception
   rc := 0;
   errormsg := 'no post_delete plguin error msg';
EXCEPTION
   WHEN others THEN
      rc := 1;
      errormsg := 'exception: post_delete plguin';
END;
END LDAP_PLUGIN_EXAMPLE2;
/
(ldapplugincontext IN  ODS.plugincontext,
result   IN  INTEGER,
dn       IN  VARCHAR2,
rc       OUT INTEGER,
errormsg OUT VARCHAR2
)
    IS
     retval       PLS_INTEGER;
     my_session   DBMS_LDAP.session;
     newDN        VARCHAR2(256);
BEGIN
   retval         := -1;
   my_session := LDAP_PLUGIN.init(ldapplugincontext);
   -- bind to the directory
   retval := LDAP_PLUGIN.simple_bind_s(ldapplugincontext, my_session);
   -- if retval is not 0, then raise exception
   newDN := REPLACE(dn, 'DIT1', 'DIT2');
   retval := DBMS_LDAP.delete_s(my_session, newDN);
   -- if retval is not 0, then raise exception
   rc := 0;
   errormsg := 'no post_delete plguin error msg';
EXCEPTION
   WHEN others THEN
      rc := 1;
      errormsg := 'exception: post_delete plguin';
END;
END LDAP_PLUGIN_EXAMPLE2;
/
```

4. Register the plug-in entry with Oracle Internet Directory.

   Construct the LDIF file `register_post_delete.ldif`:

```
cn=post_delete,cn=plugin,cn=subconfigsubentry
objectclass=orclPluginConfig
objectclass=top
orclPluginName=ldap_plugin_example2
orclPluginType=operational
orclPluginTiming=post
orclPluginLDAPOperation=ldapdelete
orclPluginEnable=1
orclPluginSubscriberDNList=cn=DIT1,cn=oraclecontext,cn=products
orclPluginVersion=1.0.1
cn=post_delete
orclPluginKind=PLSQL
```

   Use the `ldapadd` command-line tool to add this entry:

```
% ldapadd -p port_number -h host_name -D bind_dn -w passwd -v -f register_
post_delete.ldif
```

Binary Support in the Plug-in Framework

Starting with release 10.1.2, object definitions in the Plug-in LDAP API enable `ldapmodify`, `ldapadd`, and `ldapcompare` plug-ins to access binary attributes in the directory database. Formerly, only attributes of type `VARCHAR2` could be accessed. These object definitions do not invalidate plug-in code that precedes release 10.1.2. No change to this code is required. The new definitions appear in the section "Database Object Types Defined".

The section that you are reading now examines binary operations involving the three types of plug-ins. It includes examples of these plug-ins. The new object definitions apply to pre, post, and when versions of all three.

Note that the three examples use RAW functions and variables in place of LOBs.

Binary Operations with ldapmodify

The `modobj` object that the plug-in framework passes to an `ldapmodify` plug-in now holds the values of binary attributes as `binvals`. This variable is a table of `binvalobj` objects.

The plug-in determines whether a binary operation is being performed by examining the `operation` field of `modobj`. It checks whether any of the values `DBMS_LDAP.MOD_ADD`, `DBMS_LDAP.MOD_DELETE`, and `DBMS_LDAP.MOD_REPLACE` are paired with `DBMS_LDAP.MOD_BVALUES`. The pairing `DBMS_LDAP.MOD_ADD+DBMS_LDAP.MOD_BVALUES`, for example, signifies a binary add in the modify operation.

The example that follows shows a post `ldapmodify` plug-in modifying an entry in another directory. The plug-in is invoked after `ldapmodify` applies the same change to the same entry in the plug-in directory. The entry in the other directory appears under the DIT `cn=users,dc=us,dc=acme,dc=com`.

```
create or replace package moduser as
   procedure post_modify(ldapplugincontext IN ODS.plugincontext,
                         result IN integer,
                         dn IN varchar2,
                         mods IN ODS.modlist,
                         rc OUT integer,
                         errormsg OUT varchar2);
end moduser;
/
show error

CREATE OR REPLACE PACKAGE BODY moduser AS
   procedure post_modify(ldapplugincontext IN ODS.plugincontext,
                         result IN integer,
                         dn IN varchar2,
                         mods IN ODS.modlist,
                         rc OUT integer,
                         errormsg OUT varchar2)
   is
      counter1 pls_integer;
      counter2 pls_integer;
      retval pls_integer := -1;
      user_session DBMS_LDAP.session;
      user_dn varchar(256);
      user_array DBMS_LDAP.mod_array;
      user_vals DBMS_LDAP.string_collection;
      user_binvals DBMS_LDAP.blob_collection;
      ldap_host varchar(256);
```

```
   ldap_port varchar(256);
   ldap_user varchar(256);
   ldap_passwd varchar(256);
begin
   ldap_host := 'backup.us.oracle.com';
   ldap_port := '4000';
   ldap_user := 'cn=orcladmin';
   ldap_passwd := 'welcome';

   plg_debug('START MODIFYING THE ENTRY');

   -- Get a session
      user_session := dbms_ldap.init(ldap_host, ldap_port);

   -- Bind to the directory
      retval := dbms_ldap.simple_bind_s(user_session, ldap_user,
      ldap_passwd);

   -- Create a mod_array
      user_array := dbms_ldap.create_mod_array(mods.count);

   -- Create a user_dn
      user_dn := substr(dn,1,instr(dn,',',1,1))||'cn=users,dc=us,dc=acme,
      dc=com';

     plg_debug('THE CREATED DN IS '||user_dn);

   -- Iterate through the modlist
      for counter1 in 1..mods.count loop

   -- Log the attribute name and operation
      if (mods(counter1).operation > DBMS_LDAP.MOD_BVALUES) then
        plg_debug('THE NAME OF THE BINARY ATTR. IS '||mods(counter1).type);
      else
        plg_debug('THE NAME OF THE NORMAL ATTR. IS '||mods(counter1).type);
      end if;
        plg_debug('THE OPERATION IS '||mods(counter1).operation);

   -- Add the attribute values to the collection
      for counter2 in 1..mods(counter1).vals.count loop
        user_vals(counter2) := mods(counter1).vals(counter2).val;
      end loop;

   -- Add the attribute values to the collection
      for counter2 in 1..mods(counter1).binvals.count loop
        plg_debug('THE NO. OF BYTES OF THE BINARY ATTR. VALUE IS '
        ||mods(counter1).binvals(counter2).length);
        user_binvals(counter2) := mods(counter1).binvals(counter2).binval;
      end loop;

   -- Populate the mod_array accordingly with binary/normal attributes
      if (mods(counter1).operation >= DBMS_LDAP.MOD_BVALUES) then
        dbms_ldap.populate_mod_array(user_array,mods(counter1).operation -
        DBMS_LDAP.MOD_BVALUES,mods(counter1).type,user_binvals);
        user_binvals.delete;
      else
        dbms_ldap.populate_mod_array(user_array,mods(counter1).operation,
        mods(counter1).type,user_vals);
        user_vals.delete;
      end if;
```

```
        end loop;

      -- Modify the entry
         retval := dbms_ldap.modify_s(user_session,user_dn,user_array);
         if retval = 0 then
           rc := 0;
           errormsg:= 'No error occured while modifying the entry';
         else
           rc := retval;
           errormsg := 'Error code '||rc||' while modifying the entry';
         end if;

      -- Free the mod_array
         dbms_ldap.free_mod_array(user_array);

       plg_debug('FINISHED MODIFYING THE ENTRY');

   exception
 WHEN others THEN
   plg_debug (SQLERRM);
 end;
end moduser;
/
show error

exit;
```

Binary Operations with ldapadd

The `entryobj` object that the plug-in framework passes to an `ldapadd` plug-in now holds binary attributes as `binattr`. This variable is a table of `binattrobj` objects. The example that follows shows a post-add plug-in propagating a change (an added user) in the plug-in directory to another directory. In the latter directory, the entry appears under the DIT `cn=users,dc=us,dc=acme,dc=com`.

```
create or replace package adduser as
   procedure post_add(ldapplugincontext IN ODS.plugincontext,
                          result IN integer,
                          dn IN varchar2,
                          entry IN ODS.entryobj,
                          rc OUT integer,
                          errormsg OUT varchar2);
end adduser;
/
show error

CREATE OR REPLACE PACKAGE BODY adduser AS
   procedure post_add(ldapplugincontext IN ODS.plugincontext,
                          result IN integer,
                          dn IN varchar2,
                          entry IN ODS.entryobj,
                          rc OUT integer,
                          errormsg OUT varchar2)
   is
      counter1 pls_integer;
      counter2 pls_integer;
      retval pls_integer := -1;
      s integer;
      user_session DBMS_LDAP.session;
```

```
   user_dn varchar(256);
   user_array DBMS_LDAP.mod_array;
   user_vals DBMS_LDAP.string_collection;
   user_binvals DBMS_LDAP.blob_collection;
   ldap_host varchar(256);
   ldap_port varchar(256);
   ldap_user varchar(256);
   ldap_passwd varchar(256);
begin
   ldap_host := 'backup.us.oracle.com';
   ldap_port := '4000';
   ldap_user := 'cn=orcladmin';
   ldap_passwd := 'welcome';

   plg_debug('START ADDING THE ENTRY');

   -- Get a session
      user_session := dbms_ldap.init(ldap_host, ldap_port);

   -- Bind to the directory
      retval := dbms_ldap.simple_bind_s(user_session, ldap_user, ldap_passwd);

   -- Create a mod_array
      user_array := dbms_ldap.create_mod_array(entry.binattr.count +
      entry.attr.count);

   -- Create a user_dn
      user_dn := substr(dn,1,instr(dn,',',1,1))||'cn=users,dc=us,dc=acme,
      dc=com';
      plg_debug('THE CREATED DN IS '||user_dn);

   -- Populate the mod_array with binary attributes
      for counter1 in 1..entry.binattr.count loop
        for counter2 in 1..entry.binattr(counter1).binattrval.count loop
          plg_debug('THE NAME OF THE BINARY ATTR. IS '||
          entry.binattr(counter1).binattrname);
          s := dbms_lob.getlength(entry.binattr(counter1).
          binattrval(counter2));
          plg_debug('THE NO. OF BYTES OF THE BINARY ATTR. VALUE IS '||s);
          user_binvals(counter2) := entry.binattr(counter1).
          binattrval(counter2);
        end loop;
      dbms_ldap.populate_mod_array(user_array,DBMS_LDAP.MOD_ADD,
      entry.binattr(counter1).binattrname,user_binvals);
      user_binvals.delete;
     end loop;

   -- Populate the mod_array with attributes
      for counter1 in 1..entry.attr.count loop
        for counter2 in 1..entry.attr(counter1).attrval.count loop
          plg_debug('THE NORMAL ATTRIBUTE '||entry.attr(counter1).attrname||'
          HAS THE VALUE '||entry.attr(counter1).attrval(counter2));
          user_vals(counter2) := entry.attr(counter1).attrval(counter2);
        end loop;
      dbms_ldap.populate_mod_array(user_array,DBMS_LDAP.MOD_ADD,
      entry.attr(counter1).attrname,user_vals);
      user_vals.delete;
      end loop;

   -- Add the entry
```

```
        retval := dbms_ldap.add_s(user_session,user_dn,user_array);
        plg_debug('THE RETURN VALUE IS '||retval);
        if retval = 0 then
          rc := 0;
          errormsg:= 'No error occured while adding the entry';
        else
          rc := retval;
          errormsg := 'Error code '||rc||' while adding the entry';
        end if;

      -- Free the mod_array
        dbms_ldap.free_mod_array(user_array);
        retval := dbms_ldap.unbind_s(user_session);

        plg_debug('FINISHED ADDING THE ENTRY');

   exception
 WHEN others THEN
   plg_debug (SQLERRM);
 end;
end adduser;
/
show error

exit;
```

Binary Operations with ldapcompare

The `ldapcompare` plug-in can use three new overloaded module interfaces to compare binary attributes. If you want to use these interfaces to develop a plug-in package that handles both binary and nonbinary attributes, you must include two separate procedures in the package. The package name for both procedures is the same because only one `orclPluginName` can be registered in the plug-in entry.

After updating an existing plug-in package to include a procedure that compares binary attributes, reinstall the package. Recompile packages that depend on the plug-in package.

The three new interfaces look like this:

```
PROCEDURE pre_compare (ldapplugincontext IN ODS.plugincontext,
                       dn          IN VARCHAR2,
                       attrname    IN VARCHAR2,
                       attrval     IN BLOB,
                       rc          OUT INTEGER,
                       errormsg    OUT VARCHAR2 );

PROCEDURE when_compare_replace (ldapplugincontext IN ODS.plugincontext,
                                result      OUT INTEGER,
                                dn          IN VARCHAR2,
                                attrname    IN VARCHAR2,
                                attrval     IN BLOB,
                                rc          OUT INTEGER,
                                errormsg    OUT VARCHAR2 );

PROCEDURE post_compare (ldapplugincontext IN ODS.plugincontext,
                        result      IN INTEGER,
                        dn          IN VARCHAR2,
                        attrname    IN VARCHAR2,
                        attrval     IN BLOB,
```

```
                        rc          OUT INTEGER,
                        errormsg    OUT VARCHAR2 );
```

The example that follows compares a binary attribute of an entry in the plug-in directory with a binary attribute of an entry in another directory. This package replaces the compare code of the server with the compare code of the plug-in. The package handles both binary and nonbinary attributes. As such it contains two separate procedures.

```
create or replace package compareattr as
   procedure when_compare_replace(ldapplugincontext IN ODS.plugincontext,
                           result OUT integer,
                           dn IN varchar2,
                           attrname IN VARCHAR2,
                           attrval IN BLOB,
                           rc OUT integer,
                           errormsg OUT varchar2);
   procedure when_compare_replace(ldapplugincontext IN ODS.plugincontext,
                           result OUT integer,
                           dn IN varchar2,
                           attrname IN VARCHAR2,
                           attrval IN varchar2,
                           rc OUT integer,
                           errormsg OUT varchar2);
end compareattr;
/
show error

CREATE OR REPLACE PACKAGE BODY compareattr AS
   procedure when_compare_replace(ldapplugincontext IN ODS.plugincontext,
                           result OUT integer,
                           dn IN varchar2,
                           attrname IN VARCHAR2,
                           attrval IN varchar2,
                           rc OUT integer,
                           errormsg OUT varchar2)
   is
   pos            INTEGER := 2147483647;
   begin
      plg_debug('START');
      plg_debug('THE ATTRNAME IS '||attrname||' AND THE VALUE IS '||attrval);
      plg_debug('END');
      rc := 0;
      errormsg := 'No error!!!';
   exception
   WHEN others THEN
    plg_debug ('Unknown UTL_FILE Error');
   end;

   procedure when_compare_replace(ldapplugincontext IN ODS.plugincontext,
                           result OUT integer,
                           dn IN varchar2,
                           attrname IN VARCHAR2,
                           attrval IN BLOB,
                           rc OUT integer,
                           errormsg OUT varchar2)
   is
      counter pls_integer;
      retval pls_integer := -1;
      cmp_result integer;
```

```
   s integer;
   user_session DBMS_LDAP.session;
   user_entry DBMS_LDAP.message;
   user_message DBMS_LDAP.message;
   user_dn varchar(256);
   user_attrs DBMS_LDAP.string_collection;
   user_attr_name VARCHAR2(256);
   user_ber_elmt DBMS_LDAP.ber_element;
   user_vals DBMS_LDAP.blob_collection;
   ldap_host varchar(256);
   ldap_port varchar(256);
   ldap_user varchar(256);
   ldap_passwd varchar(256);
   ldap_base varchar(256);
begin
   ldap_host := 'backup.us.oracle.com';
   ldap_port := '4000';
   ldap_user := 'cn=orcladmin';
   ldap_passwd := 'welcome';
   ldap_base := dn;

   plg_debug('STARTING COMPARISON IN WHEN REPLACE PLUG-IN');

     s := dbms_lob.getlength(attrval);
     plg_debug('THE NUMBER OF BYTES OF ATTRVAL '||s);

   -- Get a session
      user_session := dbms_ldap.init(ldap_host, ldap_port);

   -- Bind to the directory
      retval := dbms_ldap.simple_bind_s(user_session, ldap_user, ldap_passwd);

   -- issue the search
      user_attrs(1) := attrname;
      retval := DBMS_LDAP.search_s(user_session, ldap_base,
                                   DBMS_LDAP.SCOPE_BASE,
                                   'objectclass=*',
                                   user_attrs,
                                   0,
                                   user_message);

   -- Get the entry in the other OID server
      user_entry := DBMS_LDAP.first_entry(user_session, user_message);

   -- Log the DN and the Attribute name
      user_dn := DBMS_LDAP.get_dn(user_session, user_entry);
      plg_debug('THE DN IS '||user_dn);
      user_attr_name := DBMS_LDAP.first_attribute(user_session,user_entry,
      user_ber_elmt);

   -- Get the values of the attribute
      user_vals := DBMS_LDAP.get_values_blob(user_session, user_entry,
      user_attr_name);

   -- Start the binary comparison between the ATTRVAL and the attribute
   -- values
      if user_vals.count > 0 then
       for counter in user_vals.first..user_vals.last loop
         cmp_result := dbms_lob.compare(user_vals(counter),attrval,
   dbms_lob.getlength(user_vals(counter)),1,1);
```

```
            if  cmp_result = 0 then
                 rc := 0;
                 -- Return LDAP_COMPARE_TRUE
                 result := 6;
                 plg_debug('THE LENGTH OF THE ATTR. '||user_attr_name||' IN THE
                 ENTRY IS '||dbms_lob.getlength(user_vals(counter)));
                 errormsg := 'NO ERROR. THE COMPARISON HAS SUCCEEDED.';
                 plg_debug(errormsg);
                 plg_debug('FINISHED COMPARISON');
                 return;
            end if;
           end loop;
         end if;

      rc := 1;
      -- Return LDAP_COMPARE_FALSE
         result := 5;
         errormsg := 'ERROR. THE COMPARISON HAS FAILED.';
         plg_debug('THE LENGTH OF THE ATTR. '||user_attr_name||' IN THE ENTRY IS '
         ||dbms_lob.getlength(user_vals(user_vals.last)));
         plg_debug(errormsg);
         plg_debug('FINISHED COMPARISON');

      -- Free user_vals
         dbms_ldap.value_free_blob(user_vals);
         exception
         WHEN others THEN
           plg_debug (SQLERRM);
         end;
      end compareattr;
      /
      show error

      exit;
```

Database Object Types Defined

This section defines the object types introduced in the Plug-in LDAP API. All of these definitions are in Oracle Directory Server database schema. Note that the API includes object types that enable plug-ins to extract binary data from the database.

```
create or replace type strCollection as TABLE of VARCHAR2(512);
/
create or replace type pluginContext as TABLE of VARCHAR2(512);
/
create or replace type attrvalType as TABLE OF VARCHAR2(4000);
/
create or replace type attrobj as object (
attrname    varchar2(2000),
attrval     attrvalType
);
/
create or replace type attrlist as table of attrobj;
/
create or replace type binattrvalType as TABLE OF BLOB;
/
create or replace type binattrobj as object (
binattrname    varchar2(2000),
binattrval     binattrvalType
);
```

```
/
create or replace type binattrlist as table of binattrobj;
/
create or replace type entryobj as object (
entryname      varchar2(2000),
attr           attrlist,
binattr        binattrlist
);
/
create or replace type entrylist as table of entryobj;
/

create or replace type bvalobj as object (
length         integer,
val            varchar2(4000)
);
/
create or replace type bvallist as table of bvalobj;
/
create or replace type binvalobj as object (
length         integer,
binval         blob
);
/
create or replace type binvallist as table of binvalobj;
/
create or replace type modobj as object (
operation      integer,
type           varchar2(256),
vals           bvallist,
binvals        binvallist
);
/
create or replace type modlist as table of modobj;
```

Specifications for Plug-in Procedures

When you use the plug-ins, you must adhere to the signature defined for each of them. Each signature is provided here.

```
PROCEDURE pre_add (ldapplugincontext IN  ODS.plugincontext,
dn             IN  VARCHAR2,
entry          IN  ODS.entryobj,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2);


PROCEDURE when_add (ldapplugincontext IN  ODS.plugincontext,
dn             IN  VARCHAR2,
entry          IN  ODS.entryobj,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2);


PROCEDURE when_add_replace (ldapplugincontext IN  ODS.plugincontext,
dn             IN  VARCHAR2,
entry          IN  ODS.entryobj,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2);
```

```
PROCEDURE post_add (ldapplugincontext IN  ODS.plugincontext,
result         IN  INTEGER,
dn             IN  VARCHAR2,
entry          IN  ODS.entryobj,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2);


PROCEDURE pre_modify (ldapplugincontext IN  ODS.plugincontext,
dn             IN  VARCHAR2,
mods           IN  ODS.modlist,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2);


PROCEDURE when_modify (ldapplugincontext IN  ODS.plugincontext,
dn             IN  VARCHAR2,
mods           IN  ODS.modlist,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2);


PROCEDURE when_modify_replace (ldapplugincontext IN  ODS.plugincontext,
dn             IN  VARCHAR2,
mods           IN  ODS.modlist,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2);


PROCEDURE post_modify (ldapplugincontext IN  ODS.plugincontext,
result         IN  INTEGER,
dn             IN  VARCHAR2,
mods           IN  ODS.modlist,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2);

PROCEDURE pre_compare (ldapplugincontext IN  ODS.plugincontext,
dn             IN  VARCHAR2,
attrname       IN  VARCHAR2,
attrval        IN  VARCHAR2,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2
);

PROCEDURE pre_compare (ldapplugincontext IN ODS.plugincontext,
dn             IN VARCHAR2,
attrname       IN VARCHAR2,
attrval        IN BLOB,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2 );

PROCEDURE when_compare_replace (ldapplugincontext IN  ODS.plugincontext,
result         OUT INTEGER,
dn             IN  VARCHAR2,
attrname       IN  VARCHAR2,
attrval        IN  VARCHAR2,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2
);
```

```
PROCEDURE when_compare_replace (ldappluginContext IN ODS.pluginContext,
result         OUT INTEGER,
dn             IN VARCHAR2,
attrname       IN VARCHAR2,
attrval        IN BLOB,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2 );

PROCEDURE post_compare (ldappluginContext IN  ODS.pluginContext,
result         IN  INTEGER,
dn             IN  VARCHAR2,
attrname       IN  VARCHAR2,
attrval        IN  VARCHAR2,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2
);

PROCEDURE post_compare (ldappluginContext IN ODS.pluginContext,
result         IN INTEGER,
dn             IN VARCHAR2,
attrname       IN VARCHAR2,
attrval        IN BLOB,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2 );

PROCEDURE pre_delete (ldappluginContext IN  ODS.pluginContext,
dn             IN  VARCHAR2,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2
);

PROCEDURE when_delete (ldappluginContext IN  ODS.pluginContext,
dn             IN  VARCHAR2,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2
);
PROCEDURE when_delete_replace (ldappluginContext IN  ODS.pluginContext,
dn             IN  VARCHAR2,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2
);

PROCEDURE post_delete (ldappluginContext IN  ODS.pluginContext,
result         IN  INTEGER,
dn             IN  VARCHAR2,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2
);

PROCEDURE pre_search (ldappluginContext IN  ODS.pluginContext,
baseDN         IN  VARCHAR2,
scope          IN  INTEGER,
filterStr      IN  VARCHAR2,
requiredAttr   IN  ODS.strCollection,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2
);
```

```
PROCEDURE post_search (ldapplugincontext IN  ODS.plugincontext,
result         IN  INTEGER,
baseDN         IN  VARCHAR2,
scope          IN  INTEGER,
filterStr      IN  VARCHAR2,
requiredAttr   IN  ODS.strCollection,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2
);

PROCEDURE pre_bind (ldapplugincontext IN ODS.plugincontext,
dn             IN  VARCHAR2,
passwd         IN  VARCHAR2,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2
);

PROCEDURE when_bind_replace (ldapplugincontext IN ODS.plugincontext,
result         OUT INTEGER,
dn             IN  VARCHAR2,
passwd         IN  VARCHAR2,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2
);

PROCEDURE post_bind (ldapplugincontext IN ODS.plugincontext,
result         IN  INTEGER,
dn             IN  VARCHAR2,
passwd         IN  VARCHAR2,
rc             OUT INTEGER,
errormsg       OUT VARCHAR2
);
```

6

Integrating with Oracle Delegated Administration Services

This chapter explains how to integrate applications with Oracle Delegated Administration Services. This Web tool enables you to more easily develop tools for administering application data in the directory.

It contains the following sections:

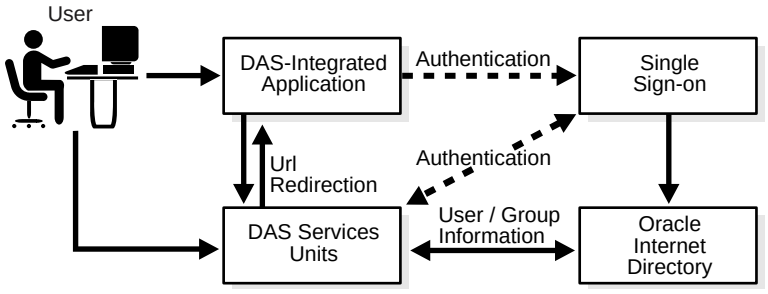
What Is Oracle Delegated Administration Services?

Oracle Delegated Administration Services consists of a set of pre-defined, Web-based service units for performing directory operations on behalf of users. These units enable directory users to update their own information.

The delegated administration services provide most of the functionality that directory-enabled applications require. You can use the service units to create user and group entries, search for entries, and change user passwords.

You can embed delegated administration service units in your applications. If, for example, you are building a Web portal, you can add service units that enable users to change application passwords stored in the directory. Each service unit has a corresponding URL stored in the directory. At runtime, an application can find the URL by querying the directory.

Figure 6–1 Overview of Delegated Administration Services



How Applications Benefit from Oracle Delegated Administration Services

An application based on Oracle Delegated Administration Services is more advanced than one based on earlier types of APIs. First, an application developed using the service units is language independent because the units are Web based. This means that the application can handle input and requests from any type of user or application, eliminating the need for a costly custom solution or configuration. Second, Oracle Delegated Administration Services comes with the Oracle Internet Directory Self-Service Console, a GUI development tool that automates many of the directory-oriented application requirements (such as Create, Edit, and Delete). Third, Oracle Delegated Administration Services is integrated with Oracle Application Server Single Sign-On. The application is automatically authenticated by the single sign-on server. This means that the application can query the directory on a user's behalf.

Integrating Applications with the Delegated Administration Services

This section contains these topics:

Integration Profile

An application integrated with Oracle Delegated Administration Services has the following characteristics:

- It is a Web-based GUI.
- It is integrated with Oracle Application Server Single Sign-On through mod_osso.
- It has operations that it must perform by way of a signed-on user. It can perform these operations using Oracle Delegated Administration Services.
- It has users or groups stored in Oracle Internet Directory and can use Oracle Delegated Administration Services for user and group management.
- It runs on the Oracle Application Server infrastructure or middle-tier. The discovery mechanism for the service URLs is inaccessible otherwise.

Oracle Delegated Administration Services Integration Methodology and Considerations

Table 6–1 on page 6-3 identifies the tasks that are required to integrate an application with Oracle Delegated Administration Services.

Table 6–1 Considerations for Integrating an Application with Oracle Delegated Administration Services

Point in Application Lifecycle	Considerations
Application design time	Examine the various services that Oracle Delegated Administration Services provides. Identify integration points within the application GUI.
	Make code changes to pass parameters to the Oracle Delegated Administration Services self-service units and to process return parameters from Oracle Delegated Administration Services.
	Introduce code in the bootstrap and installation logic to dynamically discover the location of Oracle Delegated Administration Services units from configuration information in Oracle Internet Directory. To do this, use Oracle Internet Directory Service Discovery APIs.
Application installation time	Determine the location of Oracle Delegated Administration Services units and store them in local repository.
Application runtime	Display Oracle Delegated Administration Services URLs in application GUI shown to users.
	Pass the appropriate parameters to the Oracle Delegated Administration Services by using URL encoding.
	Process return codes from Oracle Delegated Administration Services through the URL return.
Ongoing administrative activities	Provide the capability to refresh the location of Oracle Delegated Administration Services and its URLs in the administrator screens. Do this in case the deployment moves the location of Oracle Delegated Administration Services after the application has been installed.

Use Case 1: Create User

This use case shows how to integrate the Create User unit with a custom application. In the custom application page, Create User is shown as a link.

1. Identify the base URL for Oracle Delegated Administration Services by using this Java API string:

   ```
   baseUrl = Util.getDASUrl(ctx,DASURL_BASE).
   ```

   This API returns the base URL in this form: `http://host_name:port/`

2. Get the URL for the `Create User`unit by using this string:

   ```
   relUrl = Util.getDASUrl ( ctx , DASURL_CREATE_USER )
   ```

   The return value is the relative URL to access the Create User unit.

   The specific URL is the information needed to generate the link dynamically for the application.

   You can customize the parameters in Table 6–2 on page 6-4 for this unit.

Table 6–2 URL Parameters for Oracle Delegated Administration Services

Parameter	Description
homeURL	The URL that is linked to the global button **Home** in the Oracle Delegated Administration Services unit. When the calling application specifies this value, you can click **Home** to redirect the Oracle Delegated Administration Services unit to the URL specified by this parameter.
doneURL	This URL is used by Oracle Delegated Administration Services to redirect the Oracle Delegated Administration Services page at the end of each operation. In the case of Create User, once the user is created, clicking **OK** redirects the URL to this location.
cancelURL	This URL is linked with all the Cancel buttons shown in Oracle Delegated Administration Services units. Any time the user clicks Cancel, the page is redirected to the URL specified by this parameter.
enablePA	This parameter takes a Boolean value of true or false. This will enable the **Assign Privileges** section in a User or Group operation. If `enablePA` is passed with value of true in the Create User page, then the **Assign Privileges to User** section will also appear on the Create User Page.

3. Build the link with the parameters set to the following values:

```
baseUrl = http://acme.mydomain.com:7777/
relUrl =  oiddas/ui/oracle/ldap/das/admin/AppCreateUserInfoAdmin
homeURL = http://acme.mydomain.com/myapp
cancelURL = http://acme.mydomain.com/myapp
doneURL = http://acme.mydomain.com/myapp
enablePA = true
```

   The complete URL looks like this:

```
http://acme.mydomain.com:7777/oiddas/ui/oracle/ldap/das/admin/
AppCreateUserInfoAdmin?homeURL=http://acme.mydomain.com/myapp&
cancelURL=http://acme.mydomain.com/myapp&
doneURL=http://acme.mydomain.com/myapp&
enablePA=true
```

4. You can now embed this URL in the application.

Use Case 2: User LOV

List of Values (LOV) is implemented using JavaScript to invoke and pass values between the LOV calling window and the LOV page. The application invoking the LOV needs to open a popup window using JavaScript. Because Java scripts have security restrictions, no data may cross domains. Due to this limitation, only pages in the same domain can access the LOV units.

Base and relative URLs can be invoked the same way as they are for Create User. Sample files are located at:

ORACLE_HOME/ldap/das/samples/lov

The samples illustrate how the LOV can be invoked and data can be passed between the calling application and the Oracle Delegated Administration Services unit. A Complete illustration of the LOV invocation is beyond the scope of this chapter.

Java APIs Used to Access URLs

Java APIs can be used to discover URLs for Oracle Delegated Administration Services. More details about these APIs are provided in Chapter 3, "Developing Applications with Oracle Extensions to the Standard APIs" and in Chapter 12, "DAS_URL Interface Reference". The API functions that address URL discovery are `getDASUrl(DirContext ctx, String urlTypeDN)` and `getAllDASUrl(DirContext ctx)`.

7

Developing Applications for Single Sign-On

This chapter explains how to develop applications to work with mod_osso. The chapter contains the following topics:

- What Is mod_osso?
- Protecting Applications Using mod_osso: Two Methods
- Developing Applications Using mod_osso
- Security Issues: Single Sign-Off and Application Logout

What Is mod_osso?

In OracleAS release 10.1.2, you use mod_osso, an authentication module on the Oracle HTTP Server, to enable applications for single sign-on. mod_osso is a simple alternative to the single sign-on SDK, used in earlier releases to integrate partner applications. mod_osso simplifies the authentication process by serving as the sole partner application to the single sign-on server. By doing so, it renders authentication transparent for OracleAS applications.

After authenticating users, mod_osso transmits the simple header values that applications need to validate them. These include the following:

- User name
- User GUID
- Language and territory

Table 7–1 lists all of the user attributes that mod_osso passes to applications. The table also recommends attributes to use as keys, or handles, to retrieve additional user attributes from Oracle Internet Directory.

Table 7–1 User Attributes Passed to Partner Applications

HTTP Header Name	Description	Source	Use as Key or Handle?
`Osso-User-Guid`	Single sign-on user's globally unique user ID (GUID).	Single sign-on user's globally unique user ID (GUID).	Recommended.
`Osso-Subscriber-Guid`	Realm GUID.	Realm entry in Oracle Internet Directory.	Recommended.

Table 7–1 (Cont.) User Attributes Passed to Partner Applications

HTTP Header Name	Description	Source	Use as Key or Handle?
`Remote-User`	User nickname as entered by user on the login page.	Single sign-on login page.	Recommended for pre-9.0.4 applications only.
`Osso-Subscriber`	User-friendly name for a realm.	Realm entry in Oracle Internet Directory.	Not recommended. Use GUID headers to perform user searches in Oracle Internet Directory.
`Accept-Language`	Language and territory in ISO format.	Single sign-on server.	Not applicable.

mod_osso interoperates only with the Oracle HTTP listener. You can use OracleAS SSO Plug-in to protect applications that work with third-party listeners such as Sun One and IIS. To learn how to use OracleAS SSO Plug-in, see the appendix about this tool in *Oracle HTTP Server Administrator's Guide*.

Protecting Applications Using mod_osso: Two Methods

mod_osso redirects the user to the single sign-on server only if the URL you request is configured to be protected. You can secure URLs in one of two ways: statically or dynamically. Static directives simply protect the application, ceding control over user interaction to mod_osso. Dynamic directives not only protect the application, they also enable it to regulate user access.

This section contains the following topics:

- Protecting URLs Statically
- Protecting URLs with Dynamic Directives

Protecting URLs Statically

You can statically protect URLs with mod_osso by applying directives to the `mod_osso.conf` file. This file is found at *`ORACLE_HOME`*`/Apache/Apache/conf`. In the example that follows, a directory named `/private`, located just below the Oracle HTTP Server document root, is protected by this directive:

```
<IfModule mod_osso.c>

  <Location /private>
    AuthType Basic
    require valid-user
  </Location>

</IfModule>
```

After making the entry, restart the Oracle HTTP Server:

```
ORACLE_HOME/opmn/bin/opmnctl restartproc type=ohs
```

Finally, populate the directory with pages and then test them. For example:

```
http://host:port/private/helloworld.html
```

Protecting URLs with Dynamic Directives

Dynamic directives are HTTP response headers that have special error codes that enable an application to request granular functionality from the single sign-on system

without having to implement the intricacies of the single sign-on protocol. Upon receiving a directive as part of a simple HTTP response from the application, mod_osso creates the appropriate single sign-on protocol message and communicates it to the single sign-on server.

OracleAS supports dynamic directives for Java servlets and JSPs. The product does not currently support dynamic directives for PL/SQL applications.

Table 7–2 lists commonly requested dynamic directives.

Table 7–2 Commonly Requested Dynamic Directives

Directive	Status Code	Headers
Request Authentication	`401`, `499`	-
Request Forced Authentication	`499`	`Osso-Paranoid: true`
Single Sign-Off	`470`	`Osso-Return-URL` This is the URL to return to after single sign-off is complete

Developing Applications Using mod_osso

This section explains how to write and enable applications using mod_osso. The section contains the following topics:

- Developing Statically Protected PL/SQL Applications
- Developing Statically Protected Java Applications
- Developing Java Applications That Use Dynamic Directives
- A Word About Non-GET Authentication

Developing Statically Protected PL/SQL Applications

What follows is an example of a simple mod_osso-protected application. This application logs the user in to the single sign-on server, displays user information, and then logs the user out of both the application and the single sign-on server.

Use the following steps to write and enable a PL/SQL application using mod_osso.

1. Create the schema where application procedure will be loaded.

```
sqlplus sys/sys_password as sysdba
create user schema_name identified by schema_password;
grant connect,  resource to schema_name;
```

2. Load the following procedure into the schema and grant the public access to the procedure:

```
create or replace procedure show_user_info
 is
 begin
    begin
        htp.init;
     exception
         when others then null;
     end;
     htp.htmlOpen;
     htp.bodyOpen;
```

```
     htp.print('<h2>Welcome to Oracle Single Sign-On</h2>');
     htp.print('<pre>');
     htp.print('Remote user: '
        || owa_util.get_cgi_env('REMOTE_USER'));
     htp.print('User DN: '
        || owa_util.get_cgi_env('Osso-User-Dn'));
     htp.print('User Guid: '
        || owa_util.get_cgi_env('Osso-User-Guid'));
     htp.print('Subscriber: '
        || owa_util.get_cgi_env('Osso-Subscriber'));
     htp.print('Subscriber DN: '
        || owa_util.get_cgi_env('Osso-Subscriber-Dn'));
     htp.print('Subscriber Guid: '
        || owa_util.get_cgi_env('Osso-Subscriber-Guid'));
     htp.print('</pre>');
     htp.print('<a href=/osso_logout?'
        ||'p_done_url=http://my.oracle.com>Logout</a>');

     htp.bodyClose;
     htp.htmlClose;
end show_user_info;
/
show errors;

grant execute on show_user_info to public;
```

3. Create a database access descriptor (DAD) for the application in the `dads.conf` file, located at *`ORACLE_HOME`*`/Apache/modplsql/conf`:

```
<Location /pls/DAD_name>
       SetHandler pls_handler
       Order deny,allow
       AllowOverride None
       PlsqlDatabaseConnectString    hostname:port:SID
       PlsqlDatabasePassword         schema_password
       PlsqlDatabaseUsername         schema_name
       PlsqlDefaultPage              schema_name.show_user_info
       PlsqlDocumentTablename        schema_name.wwdoc_document
       PlsqlDocumentPath             docs
       PlsqlDocumentProcedure        schema_name.wwdoc_process.process_
                                     download
       PlsqlAuthenticationMode       Basic
       PlsqlPathAlias                url
       PlsqlPathAliasProcedure       schema_name.wwpth_api_alias.process_
                                     download
       PlsqlSessionCookieName        schema_name
       PlsqlCGIEnvironmentList       OSSO-USER-DN
       PlsqlCGIEnvironmentList       OSSO-USER-GUID
       PlsqlCGIEnvironmentList       OSSO-SUBSCRIBER
       PlsqlCGIEnvironmentList       OSSO-SUBSCRIBER-DN
       PlsqlCGIEnvironmentList       OSSO-SUBSCRIBER-GUID
</Location>
```

4. Protect the application DAD by entering the following lines in the `mod_osso.conf` file:

```
<Location /pls/DAD_name>
  require valid-user
  authType Basic
</Location>
```

> **Note:** The assumption here is that mod_osso is already configured for single sign-on. This step is performed when OracleAS is installed.

5. Restart the Oracle HTTP Server:

   `http://host:port/private/helloworld.html`

6. To test whether the newly created functions and procedures are protected by mod_osso, try to access them from a browser:

   `http://host:port/pls/DAD/schema_name.show_user_info`

   Selecting the URL should invoke the single sign-on login page if `mod_osso.conf` has been configured properly and mod_osso is registered with the single sign-on server.

Developing Statically Protected Java Applications

Use the following steps to write and enable a servlet or JSP application using mod_osso:

1. Write the JSP or servlet. Like the PL/SQL application example immediately preceding, the simple servlet that follows logs the user in, displays user information, and then logs the user out.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * Example servlet showing how to get the SSO User information
 */

public class SSOProtected extends HttpServlet
{

    public void service(HttpServletRequest request,
                        HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");

        // Show authenticated user informationsingle sign-on
        PrintWriter out = response.getWriter();
        out.println("<h2>Welcome to Oracle Single Sign-On</h2>");
        out.println("<pre>");
        out.println("Remote user: "
            + request.getRemoteUser());
        out.println("Osso-User-Dn: "
            +  request.getHeader("Osso-User-Dn"));
        out.println("Osso-User-Guid: "
            +  request.getHeader("Osso-User-Guid"));
        out.println("Osso-Subscriber: "
            +  request.getHeader("Osso-Subscriber"));
        out.println("Osso-User-Dn: "
            +  request.getHeader("Osso-User-Dn"));
        out.println("Osso-Subscriber-Dn: "
            +  request.getHeader("Osso-Subscriber-Dn"));
        out.println("Osso-Subscriber-Guid: "
```

```
            +  request.getHeader("Osso-Subscriber-Guid"));
        out.println("Lang/Territory: "
            + request.getHeader("Accept-Language"));
        out.println("</pre>");
        out.println("<a href=/osso_logout?"
            +"p_done_url=http://my.oracle.com>Logout</a>");
```

2. Protect the servlet by entering the following lines in the `mod_osso.conf` file:

```
<Location /servlet>
   require valid-user
   authType Basic
</Location>
```

3. Deploy the servlet. If you need help, see the overview chapter in *Oracle Application Server Containers for J2EE Servlet Developer's Guide*. This chapter provides an example of a servlet and shows how to deploy it.

4. Restart the Oracle HTTP Server and OC4J:

```
ORACLE_HOME/opmn/bin/opmnctl restartproc type=ohs
ORACLE_HOME/opmn/bin/opmnctl stopproc type=oc4j
ORACLE_HOME/opmn/bin/opmnctl startproc type=oc4j
```

5. Test the servlet by trying to access it from the browser. Selecting the URL should invoke the login page.

   The process is this: when you try to access the servlet from the browser, you are redirected to the single sign-on server for authentication. Next you are redirected back to the servlet, which displays user information. You may then select the logout link to log out of the application as well as the single sign-on server.

Developing Java Applications That Use Dynamic Directives

Applications that use dynamic directives require no entry in `mod_osso.conf` because mod_osso protection is written directly into the application as one or more dynamic directives. The servlets that follow show how such directives are incorporated. Like their "static" counterparts, these sample "dynamic" applications generate user information.

This section covers the following topics:

- Java Example #1: Simple Authentication
- Java Example #2: Single Sign-Off
- Java Example #3: Forced Authentication

Java Example #1: Simple Authentication

This servlet uses the `request.getRemoteUser()`method to check the mod_osso cookie for the user name. If the user name is absent, the servlet issues dynamic directive `499`, a request for simple authentication. The key lines are in boldface.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * Example servlet showing how to use
 * Dynamic Directive for login
 */
```

```
public class SSODynLogin extends HttpServlet
{

    public void service(HttpServletRequest request,
                        HttpServletResponse response)
        throws IOException, ServletException
    {
        String l_user     = null;

        // Try to get the authenticate user name
        try
        {
            l_user = request.getRemoteUser();
        }
        catch(Exception e)
        {
            l_user = null;
        }

        // If user is not authenticated then generate
        // dynamic directive for authentication
        if((l_user == null) || (l_user.length() <= 0) )
        {
           response.sendError(499, "Oracle SSO");
        }
        else
        {
            // Show authenticated user information
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            out.println("<h2>Welcome to Oracle Single Sign-On</h2>");
            out.println("<pre>");
            out.println("Remote user: "
                + request.getRemoteUser());
            out.println("Osso-User-Dn: "
                +  request.getHeader("Osso-User-Dn"));
            out.println("Osso-User-Guid: "
                +  request.getHeader("Osso-User-Guid"));
            out.println("Osso-Subscriber: "
                +  request.getHeader("Osso-Subscriber"));
            out.println("Osso-User-Dn: "
                +  request.getHeader("Osso-User-Dn"));
            out.println("Osso-Subscriber-Dn: "
                +  request.getHeader("Osso-Subscriber-Dn"));
            out.println("Osso-Subscriber-Guid: "
                +  request.getHeader("Osso-Subscriber-Guid"));
            out.println("Lang/Territory: "
                + request.getHeader("Accept-Language"));
            out.println("</pre>");
        }
    }
```

Note: If Oracle JAAS Provider is used, the directive code `401` may be substituted for `499`.

Java Example #2: Single Sign-Off

This servlet is invoked when users select the login link within an application. The application sets the URL to return to when sign-off is complete; then it issues a directive that sends users to the single sign-off page. The key lines are in boldface.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * Example servlet showing how to use
 * Dynamic Directive for logout
 */

public class SSODynLogout extends HttpServlet
{
    public void service (HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException
    {
        // Set the return URL
        response.setHeader("Osso-Return-Url",
               "http://my.oracle.com" );
        // Send Dynamic Directive for logout
        response.sendError(470, "Oracle SSO");
    }
}
```

> **Note:** Alternatively, you can redirect to the `osso_logout` URL on that computer.

Java Example #3: Forced Authentication

If logged-in users have exceeded a timeout, an application can force them to reauthenticate. The directive for reauthentication is written into the servlet that follows. The key lines are in boldface.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * Example servlet showing how to use
 * Dynamic Directive for forced login
 */

public class SSODynForcedLogin extends HttpServlet
{

    public void service(HttpServletRequest request,
                        HttpServletResponse response)
        throws IOException, ServletException
    {
        String l_user = null;
        // Try to get the authenticate user name
        try
        {
            l_user = request.getRemoteUser();
        }
```

```
            catch(Exception e)
            {
                l_user = null;
            }

            // If the user is authenticated then show
            // user information; otherwise generate Dynamic
            // Directive for forced login
            if(l_user != null)
            {
                // Show authenticated user information
                PrintWriter out = response.getWriter();
                response.setContentType("text/html");
                out.println("<h2>Welcome to Oracle Single Sign-On.</h2>");
                out.println("<pre>");
                out.println("Remote user: "
                    + request.getRemoteUser());
                out.println("Osso-User-Dn: "
                    +  request.getHeader("Osso-User-Dn"));
                out.println("Osso-User-Guid: "
                    +  request.getHeader("Osso-User-Guid"));
                out.println("Osso-Subscriber: "
                    +  request.getHeader("Osso-Subscriber"));
                out.println("Osso-User-Dn: "
                    +  request.getHeader("Osso-User-Dn"));
                out.println("Osso-Subscriber-Dn: "
                    +  request.getHeader("Osso-Subscriber-Dn"));
                out.println("Osso-Subscriber-Guid: "
                    +  request.getHeader("Osso-Subscriber-Guid"));
                out.println("Lang/Territory: "
                    + request.getHeader("Accept-Language"));
                out.println("</pre>");
            }
            else
            {
                response.setHeader( "Osso-Paranoid", "true" );
                response.sendError(499, "Oracle SSO");
            }
        }
    }
```

A Word About Non-GET Authentication

The first page of a mod_osso-protected application must be a URL that uses the GET authentication method. If the POST method is used, the data that the user provides when logging in is lost during redirection to the single sign-on server. When deciding whether to enable the global user inactivity timeout, please note that users are redirected after timing out and logging in again.

Security Issues: Single Sign-Off and Application Logout

If you build custom applications using OracleAS, note the following: when global logout, or single sign-off, is invoked, only the single sign-on and mod_osso cookies are cleared. This means that an OracleAS application must be coded to store single sign-on user and realm names in either the OC4J session or in the application session. The application must then compare these values to those passed by mod_osso. If a match occurs, the application must show personalized content. If no match occurs, which

means that the mod_osso cookie is absent, the application must clear the application session and force the user to log in.

This section covers the following topics:

- Application Login: Code Examples
- Application Logout: Recommended Code

Application Login: Code Examples

The first two code examples in this section do not incorporate the logic prescribed in the section immediately preceding. The third example does incorporate this logic. Although these are Java examples, they could be examples written in other languages such as Perl, PL/SQL, and CGI.

Bad Code Example #1

```
// Get user name from application session. This session was
// established by the application cookie or OC4J session cookie
String username = request.getSession().getAttribute('USER_NAME');

// Get subscriber name from application session. This session was
// established by the application cookie or OC4J session cookie.
String subscriber = request.getSession().getAttribute('SUBSCRIBER_NAME');

// Get user security information from application session. This session was
established  by the application cookie or OC4J session cookie
String user_sec_info = request.getSession().getAttribute('USER_APP_SEC');

if((username != null) && (subscriber!= null))
{
  // Show personalized user content
  show_personalized_page(username, subscriber, user_sec_info);
}
else
{
  // Send Dynamic Directive for login
  response.sendError( 499, "Oracle SSO" );
```

Bad Code Example #2

```
// Get SSO username from http header
String username = request.getRemoteUser();

// Get subscriber name from SSO http header
String subscriber = request.getHeader('OSSO-SUBSCRIBER');

// Get user security information from application session.
// This session was established by the application or OC4J session.
String user_sec_info =request.getSession().getAttribute('USER_APP_SEC');

if((ssousername != null)&&(subscriber!= null))
{
  // Show personalized user content
show_personalized_page(username, subscriber, user_sec_info);
}
else
{
  // Send Dynamic Directive for login
response.sendError( 499, "Oracle SSO" );
```

```
}
```

Recommended Code

```
// Get user name from application session. This session was
// established by the application or OC4J session
String username = request.getSession().getAttribute('USER_NAME');

// Get subscriber name from application session. This session was
// established by the application or OC4J session
String subscriber = request.getSession().getAttribute('SUBSCRIBER_NAME');

// Get user security information from application session.
 // This session was established by the application or OC4J session.
String user_sec_info = request.getSession().getAttribute('USER_APP_SEC');

// Get username and subscriber name from JAZN API */
JAZNUserAdaptor jaznuser  = (JAZNUserAdaptor)requset.getUserPrincipal();
     String ssousername   = jaznuser.getName();
     String ssosubscriber = jaznuser.getRealm().getName();

// If you are not using JAZN api then you can also get the username and
// subscriber name from mod_osso headers
String ssousername   = request.getRemoteUser();
String ssosubscriber = request.getHeader('OSSO-SUBSCRIBER');

// Check for application session. Create it if necessary.
if((username == null) || (subscriber == null) )
  {
   ...Code to create application session. Get the user information from
   JAZN api (or mod_osso headers if you are not using JAZN api) and populate the
   application session with user, subscriber, and user security info.
  }

if((username != null)&&(subscriber != null)
  &&(ssousername != null)&&(ssosubscriber != null)
  &&(username.equalsIgnoreCase(ssousername) == 0 )
  &&(subscriber.equalsIgnoreCase(ssosubscriber) == 0))
{
  // Show personalized user content
show_personalized_page(username, subscriber, user_sec_info);
}
else
{
     ...Code to Wipe-out application session, followed by...

// Send Dynamic Directive for login
// If you are using JAZN then you should use following code
// response.sendError( 401);

// If you are not using JAZN api then you should use following code
// response.sendError( 499, "Oracle SSO" );
}
```

Application Logout: Recommended Code

Most applications that authenticate users have a logout link. In a single-sign-on-enabled application, the user invokes the dynamic directive for logout in addition to other code in the logout handler of the application. Invoking the logout

directive initiates single sign-off, or global logout. The example that follows shows what single sign-off code should look like in Java.

```
// Clear application session, if any
String l_return_url := return url to your application
response.setHeader( "Osso-Return-Url", l_return_url);
response.sendError( 470, "Oracle SSO" );
```

Part II

Oracle Internet Directory Programming Reference

Part II presents the standard APIs and the Oracle extensions to these APIs. It contains these chapters:

8

C API Reference

This chapter introduces the Oracle Internet Directory C API and provides examples of how to use it.

The chapter contains these topics:

About the Oracle Internet Directory C API

The Oracle Internet Directory SDK C API is based on LDAP Version 3 C API and Oracle extensions to support SSL.

You can use the Oracle Internet Directory API 10*g* Release 2 (10.1.2) in the following modes:

- SSL—All communication secured by using SSL
- Non-SSL—Client/server communication not secure

The API uses TCP/IP to connect to a directory server. When it does this, it uses, by default, an unencrypted channel. To use the SSL mode, you must use the Oracle SSL call interface. You determine which mode you are using by the presence or absence of the SSL calls in the API usage. You can easily switch between SSL and non-SSL modes.

> **See Also:** "Sample C API Usage" on page 8-41 for more details on how to use the two modes

This section contains these topics:

Oracle Internet Directory SDK C API SSL Extensions

Oracle SSL extensions to the LDAP API are based on standard SSL protocol. The SSL extensions provide encryption and decryption of data over the wire and authentication.

There are three modes of authentication:

- None—Neither client nor server is authenticated, and only SSL encryption is used
- One-way—Only the server is authenticated by the client
- Two-way—Both the server and the client are authenticated by each other

The type of authentication is indicated by a parameter in the SSL interface call.

SSL Interface Calls

There is only one call required to enable SSL:

```
int ldap_init_SSL(Sockbuf *sb, text *sslwallet, text *sslwalletpasswd, int
sslauthmode)
```

The `ldap_init_SSL` call performs the necessary handshake between client and server using the standard SSL protocol. If the call is successful, then all subsequent communication happens over a secure connection.

Table 8–1 Arguments for SSL Interface Calls

Argument	Description
`sb`	Socket buffer handle returned by the `ldap_open` call as part of LDAP handle.
`sslwallet`	Location of the user wallet.
`sslwalletpasswd`	Password required to use the wallet.
`sslauthmode`	SSL authentication mode user wants to use. Possible values are: ■ `GSLC_SSL_NO_AUTH`—No authentication required ■ `GSLC_SSL_ONEWAY_AUTH`—Only server authentication required. ■ `GSLC_SSL_TWOWAY_AUTH`—Both server and client authentication required. A return value of `0` indicates success. A nonzero return value indicates an error. The error code can be decoded by using the function `ldap_err2string`.

Tip: "Sample C API Usage" on page 8-41

Wallet Support

depending on which authentication mode is being used, both the server and the client may require wallets to use the SSL feature. 10*g* Release 2 (10.1.2) of the API supports only the Oracle Wallet. You can create wallets by using Oracle Wallet Manager.

Functions in the C API

This section examines each of the functions and procedures in the C API. It explains their purpose and syntax. It also provides tips for using them.

The section contains the following topics:

- SASL Authentication Using Oracle Extensions
- SASL Authentication
- Working With Controls
- Closing the Session
- Performing LDAP Operations
- Abandoning an Operation
- Obtaining Results and Peeking Inside LDAP Messages
- Handling Errors and Parsing Results
- Stepping Through a List of Results
- Parsing Search Results

The Functions at a Glance

Table 8–2 lists all of the functions and procedures in the C API and briefly explains their purpose.

Table 8–2 Functions and Procedures in the C API

Function or Procedure	Description
`ber_free`	Free the memory allocated for a BerElement structure
`ldap_abandon_ext` `ldap_abandon`	Cancel an asynchronous operation

`ldap_add_ext`
`ldap_add_ext_s`
`ldap_add`
`ldap_add_s`
	Add a new entry to the directory

`ldap_compare_ext`
`ldap_compare_ext_s`
`ldap_compare`
`ldap_compare_s`
	Compare entries in the directory

`ldap_count_entries`	Count the number of entries in a chain of search results
`ldap_count_values`	Count the string values of an attribute
`ldap_count_values_len`	Count the binary values of an attribute
`ora_ldap_create_clientctx`	Create a client context and returns a handle to it.
`ora_ldap_create_cred_hdl`	Create a credential handle.

`ldap_delete_ext`
`ldap_delete_ext_s`
`ldap_delete`
`ldap_delete_s`
	Delete an entry from the directory

`ora_ldap_destroy_clientctx`	Destroy the client context.
`ora_ldap_free_cred_hdl`	Destroy the credential handle.
`ldap_dn2ufn`	Converts the name into a more user friendly format
`ldap_err2string`	Get the error message for a specific error code
`ldap_explode_dn` `ldap_explode_rdn`	Split up a distinguished name into its components

Table 8–2 (Cont.) Functions and Procedures in the C API

Function or Procedure	Description
`ldap_first_attribute`	Get the name of the first attribute in an entry
`ldap_first_entry`	Get the first entry in a chain of search results
`ora_ldap_get_cred_props`	Retrieve properties associated with credential handle.
`ldap_get_dn`	Get the distinguished name for an entry
`ldap_get_option`	Access the current value of various session-wide parameters
`ldap_get_values`	Get the string values of an attribute
`ldap_get_values_len`	Get the binary values of an attribute
`ldap_init` `ldap_open`	Open a connection to an LDAP server
`ora_ldap_init_SASL`	Perform SASL authentication
`ldap_memfree`	Free memory allocated by an LDAP API function call

`ldap_modify_ext`
`ldap_modify_ext_s`
`ldap_modify`
`ldap_modify_s`
	Modify an entry in the directory

`ldap_msgfree`	Free the memory allocated for search results or other LDAP operation results
`ldap_first_attribute` `ldap_next_attribute`	Get the name of the next attribute in an entry
`ldap_next_entry`	Get the next entry in a chain of search results
`ldap_perror` (Deprecated)	Prints the message supplied in message.
`ldap_rename` `ldap_rename_s`	Modify the RDN of an entry in the directory
`ldap_result2error` (Deprecated)	Return the error code from result message.

`ldap_result`
`ldap_msgfree`
`ldap_msgtype`
`ldap_msgid`
	Check the results of an asynchronous operation

`ldap_sasl_bind` `ldap_sasl_bind_s`	General authentication to an LDAP server

`ldap_search_ext`
`ldap_search_ext_s`
`ldap_search`
`ldap_search_s`
	Search the directory

`ldap_search_st`	Search the directory with a timeout value
`ldap_get_option` `ldap_set_option`	Set the value of these parameters
`ora_ldap_set_clientctx`	Add properties to the client context handle.
`ora_ldap_set_cred_props`	Add properties to credential handle.

Table 8–2 (Cont.) Functions and Procedures in the C API

Function or Procedure	Description

`ldap_simple_bind`
`ldap_simple_bind_s`
`ldap_sasl_bind`
`ldap_sasl_bind_s`
	Simple authentication to an LDAP server

`ldap_unbind_ext`
`ldap_unbind`
`ldap_unbind_s`
	End an LDAP session

`ldap_value_free`	Free the memory allocated for the string values of an attribute
`ldap_value_free` `ldap_value_free_len`	Free the memory allocated for the binary values of an attribute

This section lists all the calls available in the LDAP C API found in RFC 1823.

> **See Also:** The following URL for a more detailed explanation of these calls:
>
> `http://www.ietf.org`

Initializing an LDAP Session

The calls in this section initialize a session with an LDAP server.

ldap_init and ldap_open

`ldap_init()` initializes a session with an LDAP server, but does not open a connection. The server is not actually contacted until an operation is performed that requires it, allowing various options to be set after initialization. `ldap_open()` initializes a session and opens a connection. The two fulfill the same purpose and have the same syntax, but the first is preferred.

Syntax

```
LDAP *ldap_init
(
    const char     *hostname,
    int            portno
)
;
```

Parameters

Table 8–3 Parameters for Initializing an LDAP Session

Parameter	Description
`hostname`	Contains a space-separated list of host names or dotted strings representing the IP address of hosts running an LDAP server to connect to. Each host name in the list may include a port number. The two must be separated by a colon. The hosts are tried in the order listed until a successful connection occurs. Note: A suitable representation for including a literal IPv6[10] address in the host name parameter is desired, but has not yet been determined or implemented in practice.

Table 8–3 (Cont.) Parameters for Initializing an LDAP Session

Parameter	Description
portno	Contains the TCP port number to connect to. The default LDAP port of 389 can be obtained by supplying the constant LDAP_PORT. If hostname includes a port number, portno is ignored.

Usage Notes

ldap_init() and ldap_open() both return a session handle. This is a pointer to an opaque structure that must be passed to subsequent calls pertaining to the session. These routines return NULL if the session cannot be initialized. If the session cannot be initialized, check the error reporting mechanism for the operating system to see why the call failed.

Note that if you connect to an LDAPv2 server, one of the LDAP bind calls described later SHOULD be completed before other operations can be performed on the session. LDAPv3 does not require that a bind operation be completed before other operations are performed.

The calling program can set various attributes of the session by calling the routines described in the next section.

LDAP Session Handle Options

The LDAP session handle returned by ldap_init() is a pointer to an opaque data type representing an LDAP session. In RFC 1823 this data type was a structure exposed to the caller, and various fields in the structure could be set to control aspects of the session, such as size and time limits on searches.

In the interest of insulating callers from inevitable changes to this structure, these aspects of the session are now accessed through a pair of accessor functions, described in this section.

ldap_get_option and ldap_set_option

ldap_get_option() is used to access the current value of various session-wide parameters. ldap_set_option() is used to set the value of these parameters. Note that some options are read only and cannot be set; it is an error to call ldap_set_option() and attempt to set a read only option.

Note that if automatic referral following is enabled (the default), any connections created during the course of following referrals will inherit the options associated with the session that sent the original request that caused the referrals to be returned.

Syntax

```
int ldap_get_option
(
LDAP            *ld,
int             option,
void            *outvalue
)
;

int ldap_set_option
(
LDAP            *ld,
int             option,
const void      *invalue
```

```
)
;


#define LDAP_OPT_ON     ((void *)1)
#define LDAP_OPT_OFF    ((void *)0)
```

Parameters

Table 8–4 lists and describes the parameters for LDAP session handle options.

Table 8–4 Parameters for LDAP Session Handle Options

Parameters	Description
`ld`	The session handle. If this is `NULL`, a set of global defaults is accessed. New LDAP session handles created with `ldap_init()` or `ldap_open()` inherit their characteristics from these global defaults.
`option`	The name of the option being accessed or set. This parameter should be one of the constants listed and described in Table 8–5 on page 8-7. The hexadecimal value of the constant is listed in parentheses after the constant.
`outvalue`	The address of a place to put the value of the option. The actual type of this parameter depends on the setting of the option parameter. For outvalues of type `char **` and `LDAPControl **`, a copy of the data that is associated with the LDAP session `ld` is returned. Callers should dispose of the memory by calling `ldap_memfree()` or `ldap_controls_free()`, depending on the type of data returned.
`invalue`	A pointer to the value the option is to be given. The actual type of this parameter depends on the setting of the option parameter. The data associated with invalue is copied by the API implementation to allow callers of the API to dispose of or otherwise change their copy of the data after a successful call to `ldap_set_option()`. If a value passed for invalue is invalid or cannot be accepted by the implementation, `ldap_set_option()` should return `-1` to indicate an error.

Constants

Table 8–5 on page 8-7 lists and describes the constants for LDAP session handle options.

Table 8–5 Constants

Constant	Type for invalue parameter	Type for outvalue parameter	Description
`LDAP_OPT_API_INFO(0x00)`	Not applicable. Option is read only.	`LDAPAPIInfo*`	Used to retrieve some basic information about the LDAP API implementation at execution time. Applications need to be able to determine information about the particular API implementation they are using both at compile time and during execution. This option is read only and cannot be set.
`ORA_LDAP_OPT_RFRL_CACHE`	`void* (LDAP_OPT_ON` `void* (LDAP_OPT_OFF)`	`int *`	This option determines whether referral cache is enabled or not. If this option is set to `LDAP_OPT_ON`, the cache is enabled; otherwise, the cache is disabled.
`ORA_LDAP_OPT_RFRL_CACHE_SZ`	`int *`	`int *`	This option sets the size of referral cache. The size is maximum size in terms of number of bytes the cache can grow to. It is set to 1MB by default.

Table 8–5 (Cont.) Constants

Constant	Type for invalue parameter	Type for outvalue parameter	Description
LDAP_OPT_ DEREF(0x02)	int *	int *	Determines how aliases are handled during search. It should have one of the following values: LDAP_DEREF_NEVER (0x00), LDAP_DEREF SEARCHING (0x01), LDAP_DEREF_FINDING (0x02), or LDAP_DEREF_ALWAYS (0x03). The LDAP_DEREF_SEARCHING value means aliases are dereferenced during the search but not when locating the base object of the search. The LDAP_ DEREF_FINDING value means aliases are dereferenced when locating the base object but not during the search. The default value for this option is LDAP_DEREF_ NEVER.
LDAP_OPT_ SIZELIMIT(0x03)	int *	int *	A limit on the number of entries to return from a search. A value of LDAP_NO_LIMIT (0) means no limit. The default value for this option is LDAP_NO_LIMIT.
LDAP_OPT_ TIMELIMIT(0x04)	int *	int *	A limit on the number of seconds to spend on a search. A value of LDAP_NO_LIMIT (0) means no limit. This value is passed to the server in the search request only; it does not affect how long the C LDAP API implementation itself will wait locally for search results. The timeout parameter passed to ldap_search_ext_s() or ldap_result()—both of which are described later in this document—can be used to specify both a local and server side time limit. The default value for this option is LDAP_NO_LIMIT.
LDAP_OPT_ REFERRALS(0x08)	void *(LDAP_OPT_ ON) void *(LDAP_OPT_ OFF)	int *	Determines whether the LDAP library automatically follows referrals returned by LDAP servers or not. It may be set to one of the constants LDAP_OPT_ON or LDAP_ OPT_OFF. Any non-null pointer value passed to ldap_set_option() enables this option. When the current setting is read using ldap_get_option(), a zero value means off and any nonzero value means on. By default, this option is turned on.
LDAP_OPT_ RESTART(0X09)	void * (LDAP_ OPT_ON) void * (LDAP_ OPT_OFF)	int *	Determines whether LDAP input and output operations are automatically restarted if they stop prematurely. It may be set to either LDAP_OPT_ON or LDAP_ OPT_OFF. Any non-null pointer value passed to ldap_set_option() enables this option. When the current setting is read using ldap_get_option(), a zero value means off and any nonzero value means on. This option is useful if an input or output operation can be interrupted prematurely—by a timer going off, for example. By default, this option is turned off.

Table 8–5 (Cont.) Constants

Constant	Type for invalue parameter	Type for outvalue parameter	Description
LDAP_OPT_PROTOCOL_VERSION(0x11)	int *	int *	This option indicates the version of the LDAP protocol used when communicating with the primary LDAP server. The option should be either LDAP_VERSION2 (2) or LDAP_VERSION3 (3). If no version is set, the default is LDAP_VERSION2 (2).
LDAP_OPT_SERVER_CONTROLS(0x12)	LDAPControl**	LDAPControl***	A default list of LDAP server controls to be sent with each request. **See Also:** "Working With Controls" on page 8-14
LDAP_OPT_CLIENT_CONTROLS(0x13)	LDAPControl**	LDAPControl***	A default list of client controls that affect the LDAP session. **See Also:** "Working With Controls" on page 8-14
LDAP_OPT_API_FEATURE_INFO(0x15)	Not applicable. Option is read only.	LDAPAPIFeatureInfo *	Used to retrieve version information about LDAP API extended features at execution time. Applications need to be able to determine information about the particular API implementation they are using both at compile time and during execution. This option is read only. It cannot be set.
LDAP_OPT_HOST_NAME(0x30)	char *	char **	The host name (or list of hosts) for the primary LDAP server. See the definition of the hostname parameter for ldap_init() to determine the syntax.
LDAP_OPT_ERROR_NUMBER(0x31)	int *	int *	The code of the most recent LDAP error during this session.
LDAP_OPT_ERROR_STRING(0x32)	char *	-	The message returned with the most recent LDAP error during this session.
LDAP_OPT_MATCHED_DN(0x33)	char *	char **	The matched DN value returned with the most recent LDAP error during this session.

Usage Notes

Both ldap_get_option() and ldap_set_option() return 0 if successful and -1 if an error occurs. If -1 is returned by either function, a specific error code may be retrieved by calling ldap_get_option() with an option value of LDAP_OPT_ERROR_NUMBER. Note that there is no way to retrieve a more specific error code if a call to ldap_get_option() with an option value of LDAP_OPT_ERROR_NUMBER fails.

When a call to ldap_get_option() succeeds, the API implementation MUST NOT change the state of the LDAP session handle or the state of the underlying implementation in a way that affects the behavior of future LDAP API calls. When a call to ldap_get_option() fails, the only session handle change permitted is setting the LDAP error code (as returned by the LDAP_OPT_ERROR_NUMBER option).

When a call to ldap_set_option() fails, it must not change the state of the LDAP session handle or the state of the underlying implementation in a way that affects the behavior of future LDAP API calls.

Standards track documents that extend this specification and specify new options should use values for option macros that are between 0x1000 and 0x3FFF inclusive. Private and experimental extensions should use values for the option macros that are between 0x4000 and 0x7FFF inclusive. All values less than 0x1000 and greater than 0x7FFF that are not defined in this document are reserved and should not be used. The following macro must be defined by C LDAP API implementations to aid extension implementers:

```
#define LDAP_OPT_PRIVATE_EXTENSION_BASE 0x4000  /* to 0x7FFF inclusive */
```

Authenticating to the Directory

The functions in this section are used to authenticate an LDAP client to an LDAP directory server.

ldap_sasl_bind, ldap_sasl_bind_s, ldap_simple_bind, and ldap_simple_bind_s

The `ldap_sasl_bind()` and `ldap_sasl_bind_s()` functions can be used to do general and extensible authentication over LDAP through the use of the Simple Authentication Security Layer. The routines both take the DN to bind as, the method to use, as a dotted-string representation of an object identifier identifying the method, and a `struct berval` holding the credentials. The special constant value `LDAP_SASL_SIMPLE (NULL)` can be passed to request simple authentication, or the simplified routines `ldap_simple_bind()` or `ldap_simple_bind_s()` can be used.

Syntax

```
int ldap_sasl_bind
(
LDAP                    *ld,
const char              *dn,
const char              *mechanism,
const struct berval     *cred,
LDAPControl             **serverctrls,
LDAPControl             **clientctrls,
int                     *msgidp
);

int ldap_sasl_bind_s(
LDAP                    *ld,
const char              *dn,
const char              *mechanism,
const struct berval     *cred,
LDAPControl             **serverctrls,
LDAPControl             **clientctrls,
struct berval           **servercredp
);


int ldap_simple_bind(
LDAP                    *ld,
const char              *dn,
const char              *passwd
);
```

```
int ldap_simple_bind_s(
LDAP                    *ld,
const char              *dn,
const char              *passwd
);
```

The use of the following routines is deprecated and more complete descriptions can be found in RFC 1823:

- `int ldap_bind( LDAP *ld, const char *dn, const char *cred, int method );`
- `int ldap_bind_s( LDAP *ld, const char *dn, const char *cred, int method );`
- `int ldap_kerberos_bind( LDAP *ld, const char *dn );`
- `int ldap_kerberos_bind_s( LDAP *ld, const char *dn );`

Parameters

Table 8–6 lists and describes the parameters for authenticating to the directory.

Table 8–6 Parameters for Authenticating to the Directory

Parameter	Description
`ld`	The session handle
`dn`	The name of the entry to bind as
`mechanism`	Either `LDAP_SASL_SIMPLE (NULL)` to get simple authentication, or a text string identifying the SASL method
`cred`	The credentials with which to authenticate. Arbitrary credentials can be passed using this parameter. The format and content of the credentials depends on the setting of the mechanism parameter.
`passwd`	For `ldap_simple_bind()`, the password to compare to the entry's userPassword attribute
`serverctrls`	List of LDAP server controls
`clientctrls`	List of client controls
`msgidp`	This result parameter will be set to the message id of the request if the `ldap_sasl_bind()` call succeeds
`servercredp`	This result parameter will be filled in with the credentials passed back by the server for mutual authentication, if given. An allocated `berval` structure is returned that should be disposed of by calling `ber_bvfree()`. `NULL` should be passed to ignore this field.

Usage Notes

Additional parameters for the deprecated routines are not described. Interested readers are referred to RFC 1823.

The `ldap_sasl_bind()` function initiates an asynchronous bind operation and returns the constant `LDAP_SUCCESS` if the request was successfully sent, or another LDAP error code if not. If successful, `ldap_sasl_bind()` places the message id of the request in `*msgidp`. A subsequent call to `ldap_result()` can be used to obtain the result of the bind.

The `ldap_simple_bind()` function initiates a simple asynchronous bind operation and returns the message id of the operation initiated. A subsequent call to `ldap_result()`, described in, can be used to obtain the result of the bind. In case of error,

`ldap_simple_bind()` will return `-1`, setting the session error parameters in the LDAP structure appropriately.

The synchronous `ldap_sasl_bind_s()` and `ldap_simple_bind_s()` functions both return the result of the operation, either the constant `LDAP_SUCCESS` if the operation was successful, or another LDAP error code if it was not.

Note that if an LDAPv2 server is contacted, no other operations over the connection can be attempted before a bind call has successfully completed.

Subsequent bind calls can be used to re-authenticate over the same connection, and multistep SASL sequences can be accomplished through a sequence of calls to `ldap_sasl_bind()` or `ldap_sasl_bind_s()`.

> **See Also:** "Handling Errors and Parsing Results" for more information about possible errors and how to interpret them.

SASL Authentication Using Oracle Extensions

The function `ora_ldap_init_SASL()` can be used for SASL based authentication. It accepts these arguments:

- DN of the entity to be authenticated.
- SASL credential handle for the entity. (This handle can be managed using `ora_ldap_create_cred_hdl()`, `ora_ldap_set_cred_props()` and `ora_ldap_free_cred_hdl()` functions).
- SASL mechanism to be used.

This function encapsulates the SASL handshake between the client and the directory server for various standard SASL mechanisms thereby reducing the coding effort involved in establishing a SASL-based connection to the directory server.

Supported SASL mechanisms:

- DIGEST-MD5

  The SASL API supports the authentication only mode of DIGEST-MD5. The other two authentication modes addressing data privacy and data integrity are yet to be supported.

  While authenticating against Oracle Internet Directory, the DN of the user has to be normalized before it is sent across to the server. This can be done either outside the SASL API using the `ora_ldap_normalize_dn()` function before the DN is passed on to the SASL API or with the SASL API by setting the `ORA_LDAP_CRED_SASL_NORM_AUTHDN` option in SASL credentials handle using `ora_ldap_set_cred_handle()`.

- EXTERNAL:

  The SASL API and SASL implementation in Oracle Internet Directory use SSL authentication as one of the external authentication mechanisms.

  Using this mechanism requires that the SSL connection (mutual authentication mode) be established to the directory server by using the `ora_ldap_init_SSL()` function. The `ora_ldap_init_SASL()` function can then be invoked with the `mechanism` argument as `EXTERNAL`. The directory server would then authenticate the user based on the user credentials in SSL connection.

ora_ldap_create_cred_hdl, ora_ldap_set_cred_props, ora_ldap_get_cred_props, and ora_ldap_free_cred_hdl

Use these functions to create and manage SASL credential handles. The `ora_ldap_create_cred_hdl` function should be used to create a SASL credential handle of certain type based on the type of mechanism used for SASL authentication. The `ora_ldap_set_cred_props()` function can be used to add relevant credentials to the handle needed for SASL authentication. The `ora_ldap_get_cred_props()` function can be used for retrieving the properties stored in the credential handle, and the `ora_ldap_free_cred_hdl()` function should be used to destroy the handle after its use.

Syntax

```
OraLdapHandle ora_ldap_create_cred_hdl
(
        OraLdapClientCtx * clientCtx,
        int                credType
);

OraLdapHandle ora_ldap_set_cred_props
(
        OraLdapClientCtx *   clientCtx,
        OraLdapHandle        cred,
        int                  String[],
        void             *   inProperty
);
OraLdapHandle ora_ldap_get_cred_props
(
        OraLdapClientCtx *   clientCtx,
        OraLdapHandle   cred,
        int                    String[],
        void                 *   outProperty
);

OraLdapHandle ora_ldap_free_cred_hdl
(
        OraLdapClientCtx *   clientCtx,
        OraLdapHandle   cred
);
```

Parameters

Table 8–7 Parameters for Managing SASL Credentials

Parameter	Description
`clientCtx`	C API Client context. This can be managed using `ora_ldap_init_clientctx()` and `ora_ldap_free_clientctx()` functions.
`credType`	Type of credential handle specific to SASL mechanism.
`cred`	Credential handle containing SASL credentials needed for a specific SASL mechanism for SASL authentication.
`String[]`	Type of credential, which needs to be added to credential handle.
`inProperty`	One of the SASL Credentials to be stored in credential handle.
`outProperty`	One of the SASL credentials stored in credential handle.

SASL Authentication

ora_ldap_init_SASL, the lone function in this section, performs SASL authentication.

ora_ldap_init_SASL

ora_ldap_init_SASL performs authentication based on the mechanism specified as one of its input arguments.

Syntax

```
int ora_ldap_init_SASL
(
OraLdapClientCtx * clientCtx,
LDAP*ld,
char* dn,
char* mechanism,
OraLdapHandle cred,
LDAPControl**serverctrls,
LDAPControl**clientctrls
);
```

Parameters

Table 8–8 Parameters for Managing SASL Credentials

Parameter	Description
clientCtx	C API Client context. This can be managed using ora_ldap_init_clientctx() and ora_ldap_free_clientctx() functions.
ld	Ldap session handle.
dn	User DN that requires authentication.
mechanism	SASL mechanism.
cred	Credentials needed for SASL authentication.
serverctrls	List of LDAP server controls
clientctrls	List of client controls

Working With Controls

LDAPv3 operations can be extended through the use of controls. Controls can be sent to a server or returned to the client with any LDAP message. These controls are referred to as server controls.

The LDAP API also supports a client-side extension mechanism through the use of client controls. These controls affect the behavior of the LDAP API only and are never sent to a server. A common data structure is used to represent both types of controls:

```
typedef struct ldapcontrol
{
char            *ldctl_oid;
struct berval   ldctl_value;
char            ldctl_iscritical;
} LDAPControl;
```

The fields in the ldapcontrol structure are described in Table 8–9.

Table 8–9 Fields in ldapcontrol Structure

Field	Description
ldctl_oid	The control type, represented as a string.
ldctl_value	The data associated with the control (if any). To specify a zero-length value, set ldctl_value.bv_len to zero and ldctl_value.bv_val to a zero-length string. To indicate that no data is associated with the control, set ldctl_value.bv_val to NULL.
ldctl_iscritical	Indicates whether the control is critical of not. If this field is nonzero, the operation will only be carried out if the control is recognized by the server or the client. Note that the LDAP unbind and abandon operations have no server response. Clients should not mark server controls critical when used with these two operations.

Some LDAP API calls allocate an ldapcontrol structure or a NULL-terminated array of ldapcontrol structures. The following routines can be used to dispose of a single control or an array of controls:

```
void ldap_control_free( LDAPControl *ctrl );
void ldap_controls_free( LDAPControl **ctrls );
```

If the ctrl or ctrls parameter is NULL, these calls do nothing.

A set of controls that affect the entire session can be set using the ldap_set_option() function described in "ldap_get_option and ldap_set_option" on page 8-6. A list of controls can also be passed directly to some LDAP API calls such as ldap_search_ext(), in which case any controls set for the session through the use of ldap_set_option() are ignored. Control lists are represented as a NULL-terminated array of pointers to ldapcontrol structures.

Server controls are defined by LDAPv3 protocol extension documents; for example, a control has been proposed to support server-side sorting of search results.

One client control is defined in this document (described in the following section). Other client controls may be defined in future revisions of this document or in documents that extend this API.

Client-Controlled Referral Processing As described previously in "LDAP Session Handle Options" on page 8-6, applications can enable and disable automatic chasing of referrals on a session-wide basic by using the ldap_set_option() function with the LDAP_OPT_REFERRALS option. It is also useful to govern automatic referral chasing on per-request basis. A client control with an OID of 1.2.840.113556.1.4.616 exists to provide this functionality.

```
/* OID for referrals client control */
#define LDAP_CONTROL_REFERRALS              "1.2.840.113556.1.4.616"

/* Flags for referrals client control value */
#define LDAP_CHASE_SUBORDINATE_REFERRALS    0x00000020U
#define LDAP_CHASE_EXTERNAL_REFERRALS       0x00000040U
```

To create a referrals client control, the ldctl_oid field of an LDAPControl structure must be set to LDAP_CONTROL_REFERRALS ("1.2.840.113556.1.4.616") and the ldctl_value field must be set to a four-octet value that contains a set of flags.

The ldctl_value.bv_len field must always be set to 4. The ldctl_value.bv_val field must point to a four-octet integer flags value. This flags value can be set to zero to disable automatic chasing of referrals and LDAPv3 references altogether. Alternatively, the flags value can be set to the value LDAP_CHASE_SUBORDINATE_REFERRALS (0x00000020U) to indicate that only LDAPv3 search continuation references are to be automatically chased by the API implementation, to the value LDAP_CHASE_EXTERNAL_REFERRALS (0x00000040U) to indicate that only LDAPv3 referrals are to be automatically chased, or the logical OR of the two flag values (0x00000060U) to indicate that both referrals and references are to be automatically chased.

Closing the Session

Use the functions in this section to unbind from the directory, to close open connections, and to dispose of the session handle.

ldap_unbind, ldap_unbind_ext, and ldap_unbind_s

ldap_unbind_ext(), ldap_unbind(), and ldap_unbind_s() all work synchronously in the sense that they send an unbind request to the server, close all open connections associated with the LDAP session handle, and dispose of all resources associated with the session handle before returning. Note, however, that there is no server response to an LDAP unbind operation. All three of the unbind functions return LDAP_SUCCESS (or another LDAP error code if the request cannot be sent to the LDAP server). After a call to one of the unbind functions, the session handle ld is invalid and it is illegal to make any further LDAP API calls using ld.

The ldap_unbind() and ldap_unbind_s() functions behave identically. The ldap_unbind_ext() function allows server and client controls to be included explicitly, but note that since there is no server response to an unbind request there is no way to receive a response to a server control sent with an unbind request.

Syntax

```
int ldap_unbind_ext( LDAP *ld, LDAPControl **serverctrls,
LDAPControl **clientctrls );
int ldap_unbind( LDAP *ld );
int ldap_unbind_s( LDAP *ld );
```

Parameters

Table 8–10 Parameters for Closing the Session

Parameter	Description
ld	The session handle
serverctrls	List of LDAP server controls
clientctrls	List of client controls
clientctrls	

Performing LDAP Operations

Use the functions in this section to search the LDAP directory and to return a requested set of attributes for each entry matched.

ldap_search_ext, ldap_search_ext_s, ldap_search, and ldap_search_s

The `ldap_search_ext()` function initiates an asynchronous search operation and returns the constant `LDAP_SUCCESS` if the request was successfully sent, or another LDAP error code if not. If successful, `ldap_search_ext()` places the message id of the request in `*msgidp`. A subsequent call to `ldap_result()` can be used to obtain the results from the search. These results can be parsed using the result parsing routines described in detail later.

Similar to `ldap_search_ext()`, the `ldap_search()` function initiates an asynchronous search operation and returns the message id of the operation initiated. As for `ldap_search_ext()`, a subsequent call to `ldap_result()` can be used to obtain the result of the bind. In case of error, `ldap_search()` will return `-1`, setting the session error parameters in the LDAP structure appropriately.

The synchronous `ldap_search_ext_s()`, `ldap_search_s()`, and `ldap_search_st()` functions all return the result of the operation, either the constant `LDAP_SUCCESS` if the operation was successful, or another LDAP error code if it was not. Entries returned from the search, if any, are contained in the `res` parameter. This parameter is opaque to the caller. Entries, attributes, values, and so on, can be extracted by calling the parsing routines described in this section. The results contained in `res` should be freed when no longer in use by calling `ldap_msgfree()`, which is described later.

The `ldap_search_ext()` and `ldap_search_ext_s()` functions support LDAPv3 server controls, client controls, and allow varying size and time limits to be easily specified for each search operation. The `ldap_search_st()` function is identical to `ldap_search_s()` except that it takes an additional parameter specifying a local timeout for the search. The local search timeout is used to limit the amount of time the API implementation will wait for a search to complete. After the local search timeout expires, the API implementation will send an abandon operation to stop the search operation.

> **See Also:** "Handling Errors and Parsing Results" for more information about possible errors and how to interpret them.

Syntax

```
int ldap_search_ext
(
LDAP            *ld,
const char      *base,
int             scope,
const char      *filter,
char            **attrs,
int             attrsonly,
LDAPControl     **serverctrls,
LDAPControl     **clientctrls,
struct timeval  *timeout,
int             sizelimit,
int             *msgidp
);

int ldap_search_ext_s
(
LDAP            *ld,
const char      *base,
int             scope,
const char      *filter,
char            **attrs,
```

```
int             attrsonly,
LDAPControl     **serverctrls,
LDAPControl     **clientctrls,
struct timeval  *timeout,
int             sizelimit,
LDAPMessage     **res
);

int ldap_search
(
LDAP            *ld,
const char      *base,
int             scope,
const char      *filter,
char            **attrs,
int             attrsonly
);

int ldap_search_s
(
LDAP            *ld,
const char      *base,
int             scope,
const char      *filter,
char            **attrs,
int             attrsonly,
LDAPMessage     **res
);

int ldap_search_st
);

LDAP            *ld,
const char      *base,
int             scope,
const char      *filter,
char            **attrs,
int             attrsonly,
struct timeval  *timeout,
LDAPMessage     **res
);
```

Parameters

Table 8–11 lists and describes the parameters for search operations.

Table 8–11 Parameters for Search Operations

Parameter	Description
`ld`	The session handle.
`base`	The DN of the entry at which to start the search.
`scope`	One of `LDAP_SCOPE_BASE (0x00)`, `LDAP_SCOPE_ONELEVEL (0x01)`, or `LDAP_SCOPE_SUBTREE (0x02)`, indicating the scope of the search.
`filter`	A character string representing the search filter. The value `NULL` can be passed to indicate that the filter `"(objectclass=*)"` which matches all entries is to be used. Note that if the caller of the API is using LDAPv2, only a subset of the filter functionality can be successfully used.

Table 8–11 (Cont.) Parameters for Search Operations

Parameter	Description
attrs	A NULL-terminated array of strings indicating which attributes to return for each matching entry. Passing NULL for this parameter causes all available user attributes to be retrieved. The special constant string LDAP_NO_ATTRS ("1.1") may be used as the only string in the array to indicate that no attribute types are to be returned by the server. The special constant string LDAP_ALL_USER_ATTRS ("*") can be used in the attrs array along with the names of some operational attributes to indicate that all user attributes plus the listed operational attributes are to be returned.
attrsonly	A boolean value that must be zero if both attribute types and values are to be returned, and nonzero if only types are wanted.
timeout	For the ldap_search_st() function, this specifies the local search timeout value (if it is NULL, the timeout is infinite). If a zero timeout (where tv_sec and tv_usec are both zero) is passed, API implementations should return LDAP_PARAM_ERROR. For the ldap_search_ext() and ldap_search_ext_s() functions, the timeout parameter specifies both the local search timeout value and the operation time limit that is sent to the server within the search request. Passing a NULL value for timeout causes the global default timeout stored in the LDAP session handle (set by using ldap_set_option() with the LDAP_OPT_TIMELIMIT parameter) to be sent to the server with the request but an infinite local search timeout to be used. If a zero timeout (where tv_sec and tv_usec are both zero) is passed in, API implementations should return LDAP_PARAM_ERROR. If a zero value for tv_sec is used but tv_usec is nonzero, an operation time limit of 1 should be passed to the LDAP server as the operation time limit. For other values of tv_sec, the tv_sec value itself should be passed to the LDAP server.
sizelimit	For the ldap_search_ext() and ldap_search_ext_s() calls, this is a limit on the number of entries to return from the search. A value of LDAP_NO_LIMIT (0) means no limit.
res	For the synchronous calls, this is a result parameter which will contain the results of the search upon completion of the call. If no results are returned, *res is set to NULL.
serverctrls	List of LDAP server controls.
clientctrls	List of client controls.

Table 8–11 (Cont.) Parameters for Search Operations

Parameter	Description
msgidp	This result parameter will be set to the message id of the request if the ldap_search_ext() call succeeds.There are three options in the session handle ld which potentially affect how the search is performed. They are: ■ LDAP_OPT_SIZELIMIT—A limit on the number of entries to return from the search. A value of LDAP_NO_LIMIT (0) means no limit. Note that the value from the session handle is ignored when using the ldap_search_ext() or ldap_search_ext_s() functions. ■ LDAP_OPT_TIMELIMIT—A limit on the number of seconds to spend on the search. A value of LDAP_NO_LIMIT (0) means no limit. Note that the value from the session handle is ignored when using the ldap_search_ext() or ldap_search_ext_s() functions. ■ LDAP_OPT_DEREF—One of LDAP_DEREF_NEVER (0x00), LDAP_DEREF_SEARCHING (0x01), LDAP_DEREF_FINDING (0x02), or LDAP_DEREF_ALWAYS (0x03), specifying how aliases are handled during the search. The LDAP_DEREF_SEARCHING value means aliases are dereferenced during the search but not when locating the base object of the search. The LDAP_DEREF_FINDING value means aliases are dereferenced when locating the base object but not during the search.

Reading an Entry

LDAP does not support a read operation directly. Instead, this operation is emulated by a search with base set to the DN of the entry to read, scope set to LDAP_SCOPE_BASE, and filter set to "(objectclass=*)" or NULL. The attrs parameter contains the list of attributes to return.

Listing the Children of an Entry

LDAP does not support a list operation directly. Instead, this operation is emulated by a search with base set to the DN of the entry to list, scope set to LDAP_SCOPE_ONELEVEL, and filter set to "(objectclass=*)" or NULL. The parameter attrs contains the list of attributes to return for each child entry.

ldap_compare_ext, ldap_compare_ext_s, ldap_compare, and ldap_compare_s

Use these routines to compare an attribute value assertion against an LDAP entry.

The ldap_compare_ext() function initiates an asynchronous compare operation and returns the constant LDAP_SUCCESS if the request was successfully sent, or another LDAP error code if not. If successful, ldap_compare_ext() places the message id of the request in *msgidp. A subsequent call to ldap_result() can be used to obtain the result of the compare.

Similar to ldap_compare_ext(), the ldap_compare() function initiates an asynchronous compare operation and returns the message id of the operation initiated. As for ldap_compare_ext(), a subsequent call to ldap_result() can be used to obtain the result of the bind. In case of error, ldap_compare() will return -1, setting the session error parameters in the LDAP structure appropriately.

The synchronous ldap_compare_ext_s() and ldap_compare_s() functions both return the result of the operation, either the constant LDAP_SUCCESS if the operation was successful, or another LDAP error code if it was not.

The `ldap_compare_ext()` and `ldap_compare_ext_s()` functions support LDAPv3 server controls and client controls.

> **See Also:** "Handling Errors and Parsing Results" for more information about possible errors and how to interpret them.

Syntax

```
int ldap_compare_ext
(
LDAP                    *ld,
const char              *dn,
const char              *attr,
const struct berval     *bvalue,
LDAPControl             **serverctrls,
LDAPControl             **clientctrls,
int                     *msgidp
);

int ldap_compare_ext_s
(
LDAP                    *ld,
const char              *dn,
const char              *attr,
const struct berval     *bvalue,
LDAPControl             **serverctrls,
LDAPControl             **clientctrls
);

int ldap_compare
(
LDAP                    *ld,
const char              *dn,
const char              *attr,
const char              *value
);
int ldap_compare_s
(
LDAP                    *ld,
const char              *dn,
const char              *attr,
const char              *value
);
```

Parameters

Table 8–12 lists and describes the parameters for compare operations.

Table 8–12 Parameters for Compare Operations

Parameter	Description
`ld`	The session handle.
`dn`	The name of the entry to compare against.
`attr`	The attribute to compare against.
`bvalue`	The attribute value to compare against those found in the given entry. This parameter is used in the extended routines and is a pointer to a `struct berval` so it is possible to compare binary values.

Table 8–12 (Cont.) Parameters for Compare Operations

Parameter	Description
value	A string attribute value to compare against, used by the ldap_compare() and ldap_compare_s() functions. Use ldap_compare_ext() or ldap_compare_ext_s() if you need to compare binary values.
serverctrls	List of LDAP server controls.
clientctrls	List of client controls.
msgidp	This result parameter will be set to the message id of the request if the ldap_compare_ext() call succeeds.

ldap_modify_ext, ldap_modify_ext_s, ldap_modify, and ldap_modify_s

Use these routines to modify an existing LDAP entry.

The ldap_modify_ext() function initiates an asynchronous modify operation and returns the constant LDAP_SUCCESS if the request was successfully sent, or another LDAP error code if not. If successful, ldap_modify_ext() places the message id of the request in *msgidp. A subsequent call to ldap_result() can be used to obtain the result of the modify.

Similar to ldap_modify_ext(), the ldap_modify() function initiates an asynchronous modify operation and returns the message id of the operation initiated. As for ldap_modify_ext(), a subsequent call to ldap_result() can be used to obtain the result of the modify. In case of error, ldap_modify() will return -1, setting the session error parameters in the LDAP structure appropriately.

The synchronous ldap_modify_ext_s() and ldap_modify_s() functions both return the result of the operation, either the constant LDAP_SUCCESS if the operation was successful, or another LDAP error code if it was not.

The ldap_modify_ext() and ldap_modify_ext_s() functions support LDAPv3 server controls and client controls.

> **See Also:** "Handling Errors and Parsing Results" for more information about possible errors and how to interpret them

Syntax

```
typedef struct ldapmod
{
int             mod_op;
char            *mod_type;
union mod_vals_u
  {
  char            **modv_strvals;
  struct berval   **modv_bvals;
  } mod_vals;
} LDAPMod;
#define mod_values      mod_vals.modv_strvals
#define mod_bvalues     mod_vals.modv_bvals

int ldap_modify_ext
(
LDAP            *ld,
const char      *dn,
LDAPMod         **mods,
LDAPControl     **serverctrls,
```

```
LDAPControl     **clientctrls,
int             *msgidp
);

int ldap_modify_ext_s
(
LDAP            *ld,
const char      *dn,
LDAPMod         **mods,
LDAPControl     **serverctrls,
LDAPControl     **clientctrls
);

int ldap_modify
(
LDAP            *ld,
const char      *dn,
LDAPMod         **mods
);

int ldap_modify_s
(
LDAP            *ld,
const char      *dn,
LDAPMod         **mods
);
```

Parameters

Table 8–13 lists and describes the parameters for modify operations.

Table 8–13 Parameters for Modify Operations

Parameter	Description
`ld`	The session handle
`dn`	The name of the entry to modify
`mods`	A `NULL`-terminated array of modifications to make to the entry
`serverctrls`	List of LDAP server controls
`clientctrls`	List of client controls
`msgidp`	This result parameter will be set to the message id of the request if the `ldap_modify_ext()` call succeeds

Table 8–14 lists and describes the fields in the LDAPMod structure.

Table 8–14 Fields in LDAPMod Structure

Field	Description
`mod_op`	The modification operation to perform. It must be one of `LDAP_MOD_ADD (0x00)`, `LDAP_MOD_DELETE (0x01)`, or `LDAP_MOD_REPLACE (0x02)`. This field also indicates the type of values included in the mod_vals union. It is logically ORed with `LDAP_MOD_BVALUES (0x80)` to select the `mod_bvalues` form. Otherwise, the `mod_values` form is used.
`mod_type`	The type of the attribute to modify.

Table 8–14 (Cont.) Fields in LDAPMod Structure

Field	Description
mod_vals	The values (if any) to add, delete, or replace. Only one of the mod_values or mod_bvalues variants can be used, selected by ORing the mod_op field with the constant LDAP_MOD_BVALUES. mod_values is a NULL-terminated array of zero-terminated strings and mod_bvalues is a NULL-terminated array of berval structures that can be used to pass binary values such as images.

Usage Notes

For LDAP_MOD_ADD modifications, the given values are added to the entry, creating the attribute if necessary.

For LDAP_MOD_DELETE modifications, the given values are deleted from the entry, removing the attribute if no values remain. If the entire attribute is to be deleted, the mod_vals field can be set to NULL.

For LDAP_MOD_REPLACE modifications, the attribute will have the listed values after the modification, having been created if necessary, or removed if the mod_vals field is NULL. All modifications are performed in the order in which they are listed.

ldap_rename and ldap_rename_s

Use these routines to change the name of an entry.

The ldap_rename() function initiates an asynchronous modify DN operation and returns the constant LDAP_SUCCESS if the request was successfully sent, or another LDAP error code if not. If successful, ldap_rename() places the DN message id of the request in *msgidp. A subsequent call to ldap_result() can be used to obtain the result of the rename.

The synchronous ldap_rename_s() returns the result of the operation, either the constant LDAP_SUCCESS if the operation was successful, or another LDAP error code if it was not.

The ldap_rename() and ldap_rename_s() functions both support LDAPv3 server controls and client controls.

> **See Also:** "Handling Errors and Parsing Results" for more information about possible errors and how to interpret them

Syntax

```
int ldap_rename
(
LDAP            *ld,
const char      *dn,
const char      *newrdn,
const char      *newparent,
int             deleteoldrdn,
LDAPControl     **serverctrls,
LDAPControl     **clientctrls,
int             *msgidp
);

int ldap_rename_s
(
LDAP            *ld,
const char      *dn,
```

```
const char      *newrdn,
const char      *newparent,
int             deleteoldrdn,
LDAPControl     **serverctrls,
LDAPControl     **clientctrls
);
```

The use of the following routines is deprecated and more complete descriptions can be found in RFC 1823:

```
int ldap_modrdn
(
LDAP            *ld,
const char      *dn,
const char      *newrdn
);

int ldap_modrdn_s
(
LDAP            *ld,
const char      *dn,
const char      *newrdn
);

int ldap_modrdn2
(
LDAP            *ld,
const char      *dn,
const char      *newrdn,
int             deleteoldrdn
);

int ldap_modrdn2_s
(
LDAP            *ld,
const char      *dn,
const char      *newrdn,
int             deleteoldrdn
);
```

Parameters

Table 8–15 lists and describes the parameters for rename operations.

Table 8–15 Parameters for Rename Operations

Parameter	Description
`ld`	The session handle.
`dn`	The name of the entry whose DN is to be changed.
`newrdn`	The new RDN to give the entry.
`newparent`	The new parent, or superior entry. If this parameter is `NULL`, only the RDN of the entry is changed. The root DN should be specified by passing a zero length string, `""`. The newparent parameter should always be `NULL` when using version 2 of the LDAP protocol; otherwise the server's behavior is undefined.

Table 8–15 (Cont.) Parameters for Rename Operations

Parameter	Description
`deleteoldrdn`	This parameter only has meaning on the rename routines if newrdn is different than the old RDN. It is a boolean value, if nonzero indicating that the old RDN value is to be removed, if zero indicating that the old RDN value is to be retained as non-distinguished values of the entry.
`serverctrls`	List of LDAP server controls.
`clientctrls`	List of client controls.
`msgidp`	This result parameter will be set to the message id of the request if the `ldap_rename()` call succeeds.

ldap_add_ext, ldap_add_ext_s, ldap_add, and ldap_add_s

Use these functions to add entries to the LDAP directory.

The `ldap_add_ext()` function initiates an asynchronous add operation and returns the constant `LDAP_SUCCESS` if the request was successfully sent, or another LDAP error code if not. If successful, `ldap_add_ext()` places the message id of the request in `*msgidp`. A subsequent call to `ldap_result()` can be used to obtain the result of the add.

Similar to `ldap_add_ext()`, the `ldap_add()` function initiates an asynchronous add operation and returns the message id of the operation initiated. As for `ldap_add_ext()`, a subsequent call to `ldap_result()` can be used to obtain the result of the add. In case of error, `ldap_add()` will return `-1`, setting the session error parameters in the LDAP structure appropriately.

The synchronous `ldap_add_ext_s()` and `ldap_add_s()` functions both return the result of the operation, either the constant `LDAP_SUCCESS` if the operation was successful, or another LDAP error code if it was not.

The `ldap_add_ext()` and `ldap_add_ext_s()` functions support LDAPv3 server controls and client controls.

> **See Also:** "Handling Errors and Parsing Results" for more information about possible errors and how to interpret them.

Syntax

```
int ldap_add_ext
(
LDAP            *ld,
const char      *dn,
LDAPMod         **attrs,
LDAPControl     **serverctrls,
LDAPControl     **clientctrls,
int             *msgidp
);

int ldap_add_ext_s
(
LDAP            *ld,
const char      *dn,
LDAPMod         **attrs,
LDAPControl     **serverctrls,
LDAPControl     **clientctrls
);
```

```
int ldap_add
(
LDAP            *ld,
const char      *dn,
LDAPMod         **attrs
);

int ldap_add_s
(
LDAP            *ld,
const char      *dn,
LDAPMod         **attrs
);
```

Parameters

Table 8–16 lists and describes the parameters for add operations.

Table 8–16 Parameters for Add Operations

Parameter	Description
`ld`	The session handle.
`dn`	The name of the entry to add.
`attrs`	The entry attributes, specified using the `LDAPMod` structure defined for `ldap_modify()`. The mod_type and `mod_vals` fields must be filled in. The `mod_op` field is ignored unless `OR`ed with the constant `LDAP_MOD_BVALUES`, used to select the `mod_bvalues` case of the `mod_vals` union.
`serverctrls`	List of LDAP server controls.
`clientctrls`	List of client controls.
`msgidp`	This result parameter will be set to the message id of the request if the `ldap_add_ext()` call succeeds.

Usage Notes

Note that the parent of the entry being added must already exist or the parent must be empty—that is, equal to the root DN—for an add to succeed.

ldap_delete_ext, ldap_delete_ext_s, ldap_delete, and ldap_delete_s

Use these functions to delete a leaf entry from the LDAP directory.

The `ldap_delete_ext()` function initiates an asynchronous delete operation and returns the constant `LDAP_SUCCESS` if the request was successfully sent, or another LDAP error code if not. If successful, `ldap_delete_ext()` places the message id of the request in `*msgidp`. A subsequent call to `ldap_result()` can be used to obtain the result of the delete.

Similar to `ldap_delete_ext()`, the `ldap_delete()` function initiates an asynchronous delete operation and returns the message id of the operation initiated. As for `ldap_delete_ext()`, a subsequent call to `ldap_result()` can be used to obtain the result of the delete. In case of error, `ldap_delete()` will return `-1`, setting the session error parameters in the LDAP structure appropriately.

The synchronous `ldap_delete_ext_s()` and `ldap_delete_s()` functions both return the result of the operation, either the constant `LDAP_SUCCESS` if the operation was successful, or another LDAP error code if it was not.

The `ldap_delete_ext()` and `ldap_delete_ext_s()` functions support LDAPv3 server controls and client controls.

> **See Also:** "Handling Errors and Parsing Results" for more information about possible errors and how to interpret them.

Syntax

```
int ldap_delete_ext
(
LDAP            *ld,
const char      *dn,
LDAPControl     **serverctrls,
LDAPControl     **clientctrls,
int             *msgidp
);

int ldap_delete_ext_s
(
LDAP            *ld,
const char      *dn,
LDAPControl     **serverctrls,
LDAPControl     **clientctrls
);
```

int ldap_delete

```
(
LDAP            *ld,
const char      *dn
);

int ldap_delete_s
(
LDAP            *ld,
const char      *dn
);
```

Parameters

Table 8–17 lists and describes the parameters for delete operations.

Table 8–17 Parameters for Delete Operations

Parameter	Description
`ld`	The session handle.
`dn`	The name of the entry to delete.
`serverctrls`	List of LDAP server controls.
`clientctrls`	List of client controls.
`msgidp`	This result parameter will be set to the message id of the request if the `ldap_delete_ext()` call succeeds.

Usage Notes

Note that the entry to delete must be a leaf entry—that is, it must have no children. Deletion of entire subtrees in a single operation is not supported by LDAP.

ldap_extended_operation and ldap_extended_operation_s

These routines enable extended LDAP operations to be passed to the server, providing a general protocol extensibility mechanism.

The `ldap_extended_operation()`function initiates an asynchronous extended operation and returns the constant `LDAP_SUCCESS` if the request was successfully sent, or another LDAP error code if not. If successful, `ldap_extended_operation()` places the message id of the request in `*msgidp`. A subsequent call to `ldap_result()` can be used to obtain the result of the extended operation which can be passed to `ldap_parse_extended_result()` to obtain the OID and data contained in the response.

The synchronous `ldap_extended_operation_s()` function returns the result of the operation, either the constant `LDAP_SUCCESS` if the operation was successful, or another LDAP error code if it was not. The `retoid` and `retdata` parameters are filled in with the OID and data from the response. If no OID or data was returned, these parameters are set to `NULL`.

The `ldap_extended_operation()` and `ldap_extended_operation_s()` functions both support LDAPv3 server controls and client controls.

> **See Also:** "Handling Errors and Parsing Results" for more information about possible errors and how to interpret them

Syntax

```
int ldap_extended_operation
(
LDAP                    *ld,
const char              *requestoid,
const struct berval     *requestdata,
LDAPControl             **serverctrls,
LDAPControl             **clientctrls,
int                     *msgidp
);

int ldap_extended_operation_s
(
LDAP                    *ld,
const char              *requestoid,
const struct berval     *requestdata,
LDAPControl             **serverctrls,
LDAPControl             **clientctrls,
char                    **retoidp,
struct berval           **retdatap
);
```

Parameters

Table 8–18 lists and describes the parameters for extended operations.

Table 8–18 Parameters for Extended Operations

Parameter	Description
`ld`	The session handle
`requestoid`	The dotted-OID text string naming the request
`requestdata`	The arbitrary data needed by the operation (if `NULL`, no data is sent to the server)

Table 8–18 (Cont.) Parameters for Extended Operations

Parameter	Description
serverctrls	List of LDAP server controls
clientctrls	List of client controls
msgidp	This result parameter will be set to the message id of the request if the ldap_extended_operation() call succeeds.
retoidp	Pointer to a character string that will be set to an allocated, dotted-OID text string returned by the server. This string should be disposed of using the ldap_memfree() function. If no OID was returned, *retoidp is set to NULL.
retdatap	Pointer to a berval structure pointer that will be set an allocated copy of the data returned by the server. This struct berval should be disposed of using ber_bvfree(). If no data is returned, *retdatap is set to NULL.

Abandoning an Operation

Use the functions in this section to abandon an operation in progress:

ldap_abandon_ext and ldap_abandon

ldap_abandon_ext() abandons the operation with message id msgid and returns the constant LDAP_SUCCESS if the abandon was successful or another LDAP error code if not.

ldap_abandon() is identical to ldap_abandon_ext() except that it does not accept client or server controls and it returns zero if the abandon was successful, -1 otherwise.

After a successful call to ldap_abandon() or ldap_abandon_ext(), results with the given message id are never returned from a subsequent call to ldap_result(). There is no server response to LDAP abandon operations.

Syntax

```
int ldap_abandon_ext
(
LDAP            *ld,
int             msgid,
LDAPControl     **serverctrls,
LDAPControl     **clientctrls
);

int ldap_abandon
(
LDAP            *ld,
int             msgid
);
```

Parameters

Table 8–19 lists and describes the parameters for abandoning an operation.

Table 8–19 Parameters for Abandoning an Operation

Parameter	Description
ld	The session handle.

Table 8–19 (Cont.) Parameters for Abandoning an Operation

Parameter	Description
msgid	The message id of the request to be abandoned.
serverctrls	List of LDAP server controls.
clientctrls	List of client controls.

See Also: "Handling Errors and Parsing Results" for more information about possible errors and how to interpret them

Obtaining Results and Peeking Inside LDAP Messages

Use the functions in this section to return the result of an operation initiated asynchronously. They identify messages by type and by ID.

ldap_result, ldap_msgtype, and ldap_msgid

`ldap_result()` is used to obtain the result of a previous asynchronously initiated operation. Note that depending on how it is called, `ldap_result()` can actually return a list or "chain" of result messages. The `ldap_result()` function only returns messages for a single request, so for all LDAP operations other than search only one result message is expected; that is, the only time the "result chain" can contain more than one message is if results from a search operation are returned.

Once a chain of messages has been returned to the caller, it is no longer tied in any caller-visible way to the LDAP request that produced it. Therefore, a chain of messages returned by calling `ldap_result()` or by calling a synchronous search routine will never be affected by subsequent LDAP API calls (except for `ldap_msgfree()` which is used to dispose of a chain of messages).

`ldap_msgfree()` frees the result messages (possibly an entire chain of messages) obtained from a previous call to `ldap_result()` or from a call to a synchronous search routine.

`ldap_msgtype()` returns the type of an LDAP message. `ldap_msgid()` returns the message ID of an LDAP message.

Syntax

```
int ldap_result
(
LDAP            *ld,
int             msgid,
int             all,
struct timeval  *timeout,
LDAPMessage     **res
);
int ldap_msgfree( LDAPMessage *res );
int ldap_msgtype( LDAPMessage *res );
int ldap_msgid( LDAPMessage *res );
```

Parameters

Table 8–20 on page 8-32 lists and describes the parameters for obtaining results and peeling inside LDAP messages.

Table 8–20 Parameters for Obtaining Results and Peeking Inside LDAP Messages

Parameter	Description
ld	The session handle.
msgid	The message id of the operation whose results are to be returned, the constant LDAP_RES_UNSOLICITED (0) if an unsolicited result is desired, or the constant LDAP_RES_ANY (-1) if any result is desired.
all	Specifies how many messages will be retrieved in a single call to ldap_result(). This parameter only has meaning for search results. Pass the constant LDAP_MSG_ONE (0x00) to retrieve one message at a time. Pass LDAP_MSG_ALL (0x01) to request that all results of a search be received before returning all results in a single chain. Pass LDAP_MSG_RECEIVED (0x02) to indicate that all messages retrieved so far are to be returned in the result chain.
timeout	A timeout specifying how long to wait for results to be returned. A NULL value causes ldap_result() to block until results are available. A timeout value of zero seconds specifies a polling behavior.
res	For ldap_result(), a result parameter that will contain the result of the operation. If no results are returned, *res is set to NULL. For ldap_msgfree(), the result chain to be freed, obtained from a previous call to ldap_result(), ldap_search_s(), or ldap_search_st(). If res is NULL, nothing is done and ldap_msgfree() returns zero.

Usage Notes

Upon successful completion, ldap_result() returns the type of the first result returned in the res parameter. This will be one of the following constants.

LDAP_RES_BIND (0x61)

LDAP_RES_SEARCH_ENTRY (0x64)

LDAP_RES_SEARCH_REFERENCE (0x73)-- new in LDAPv3

LDAP_RES_SEARCH_RESULT (0x65)

LDAP_RES_MODIFY (0x67)

LDAP_RES_ADD (0x69)

LDAP_RES_DELETE (0x6B)

LDAP_RES_MODDN (0x6D)

LDAP_RES_COMPARE (0x6F)

LDAP_RES_EXTENDED (0x78) -- new in LDAPv3

ldap_result() returns 0 if the timeout expired and -1 if an error occurs, in which case the error parameters of the LDAP session handle will be set accordingly.

ldap_msgfree() frees each message in the result chain pointed to by res and returns the type of the last message in the chain. If res is NULL, then nothing is done and the value zero is returned.

ldap_msgtype() returns the type of the LDAP message it is passed as a parameter. The type will be one of the types listed previously, or -1 on error.

ldap_msgid() returns the message ID associated with the LDAP message passed as a parameter, or -1 on error.

Handling Errors and Parsing Results

Use the functions in this section to extract information from results and to handle errors returned by other LDAP API routines.

ldap_parse_result, ldap_parse_sasl_bind_result, ldap_parse_extended_result, and ldap_err2string

Note that `ldap_parse_sasl_bind_result()` and `ldap_parse_extended_result()` must typically be used in addition to `ldap_parse_result()` to retrieve all the result information from SASL Bind and Extended Operations respectively.

The `ldap_parse_result()`, `ldap_parse_sasl_bind_result()`, and `ldap_parse_extended_result()` functions all skip over messages of type `LDAP_RES_SEARCH_ENTRY` and `LDAP_RES_SEARCH_REFERENCE` when looking for a result message to parse. They return the constant `LDAP_SUCCESS` if the result was successfully parsed and another LDAP error code if not. Note that the LDAP error code that indicates the outcome of the operation performed by the server is placed in the errcodep `ldap_parse_result()` parameter. If a chain of messages that contains more than one result message is passed to these routines they always operate on the first result in the chain.

`ldap_err2string()` is used to convert a numeric LDAP error code, as returned by `ldap_parse_result()`, `ldap_parse_sasl_bind_result()`, `ldap_parse_extended_result()` or one of the synchronous API operation calls, into an informative zero-terminated character string message describing the error. It returns a pointer to static data.

Syntax

```
int ldap_parse_result
(
LDAP            *ld,
LDAPMessage     *res,
int             *errcodep,
char            **matcheddnp,
char            **errmsgp,
char            ***referralsp,
LDAPControl     ***serverctrlsp,
int             freeit
);


int ldap_parse_sasl_bind_result
(
LDAP            *ld,
LDAPMessage     *res,
struct berval   **servercredp,
int             freeit
);


int ldap_parse_extended_result
(
LDAP            *ld,
LDAPMessage     *res,
char            **retoidp,
struct berval   **retdatap,
int             freeit
);
#define LDAP_NOTICE_OF_DISCONNECTION    "1.3.6.1.4.1.1466.20036"
```

```
char *ldap_err2string( int err );
```

The routines immediately following are deprecated. To learn more about them, see RFC 1823.

```
int ldap_result2error
(
LDAP            *ld,
LDAPMessage     *res,
int             freeit
);
void ldap_perror( LDAP *ld, const char *msg );
```

Parameters

Table 8–21 lists and describes parameters for handling errors and parsing results.

Table 8–21 Parameters for Handling Errors and Parsing Results

Parameter	Description
`ld`	The session handle.
`res`	The result of an LDAP operation as returned by `ldap_result()` or one of the synchronous API operation calls.
`errcodep`	This result parameter will be filled in with the LDAP error code field from the `LDAPMessage` message. This is the indication from the server of the outcome of the operation. `NULL` should be passed to ignore this field.
`matcheddnp`	In the case of a return of `LDAP_NO_SUCH_OBJECT`, this result parameter will be filled in with a DN indicating how much of the name in the request was recognized. `NULL` should be passed to ignore this field. The matched DN string should be freed by calling `ldap_memfree()` which is described later in this document.
`errmsgp`	This result parameter will be filled in with the contents of the error message field from the `LDAPMessage` message. The error message string should be freed by calling `ldap_memfree()` which is described later in this document. `NULL` should be passed to ignore this field.
`referralsp`	This result parameter will be filled in with the contents of the referrals field from the `LDAPMessage` message, indicating zero or more alternate LDAP servers where the request is to be retried. The referrals array should be freed by calling `ldap_value_free()` which is described later in this document. `NULL` should be passed to ignore this field.
`serverctrlsp`	This result parameter will be filled in with an allocated array of controls copied out of the `LDAPMessage` message. The control array should be freed by calling `ldap_controls_free()` which was described earlier.
`freeit`	A Boolean that determines whether the `res` parameter is disposed of or not. Pass any nonzero value to have these routines free `res` after extracting the requested information. This is provided as a convenience; you can also use `ldap_msgfree()` to free the result later. If `freeit` is nonzero, the entire chain of messages represented by res is disposed of.
`servercredp`	For SASL bind results, this result parameter will be filled in with the credentials passed back by the server for mutual authentication, if given. An allocated `berval` structure is returned that should be disposed of by calling `ber_bvfree()`. `NULL` should be passed to ignore this field.

Table 8–21 (Cont.) Parameters for Handling Errors and Parsing Results

Parameter	Description
retoidp	For extended results, this result parameter will be filled in with the dotted-OID text representation of the name of the extended operation response. This string should be disposed of by calling ldap_memfree(). NULL should be passed to ignore this field. The LDAP_NOTICE_OF_DISCONNECTION macro is defined as a convenience for clients that wish to check an OID to see if it matches the one used for the unsolicited Notice of Disconnection (defined in RFC 2251[2] section 4.4.1).
retdatap	For extended results, this result parameter will be filled in with a pointer to a struct berval containing the data in the extended operation response. It should be disposed of by calling ber_bvfree(). NULL should be passed to ignore this field.
err	For ldap_err2string(), an LDAP error code, as returned by ldap_parse_result() or another LDAP API call.

Usage Notes

See RFC 1823 for a description of parameters peculiar to the deprecated routines.

Stepping Through a List of Results

Use the routines in this section to step through the list of messages in a result chain returned by ldap_result().

ldap_first_message and ldap_next_message

The result chain for search operations can include referral messages, entry messages, and result messages.

ldap_count_messages() is used to count the number of messages returned. The ldap_msgtype() function, described previously, can be used to distinguish between the different message types.

```
LDAPMessage *ldap_first_message( LDAP *ld, LDAPMessage *res );
LDAPMessage *ldap_next_message( LDAP *ld, LDAPMessage *msg );
int ldap_count_messages( LDAP *ld, LDAPMessage *res );
```

Parameters

Table 8–22 lists and describes the parameters for stepping through a list of results.

Table 8–22 Parameters for Stepping Through a List of Results

Parameter	Description
ld	The session handle.
res	The result chain, as obtained by a call to one of the synchronous search routines or ldap_result().
msg	The message returned by a previous call to ldap_first_message() or ldap_next_message().

Usage Notes

ldap_first_message() and ldap_next_message() will return NULL when no more messages exist in the result set to be returned. NULL is also returned if an error

occurs while stepping through the entries, in which case the error parameters in the session handle ld will be set to indicate the error.

If successful, `ldap_count_messages()` returns the number of messages contained in a chain of results; if an error occurs such as the res parameter being invalid, `-1` is returned. The `ldap_count_messages()` call can also be used to count the number of messages that remain in a chain if called with a message, entry, or reference returned by `ldap_first_message()`, `ldap_next_message()`, `ldap_first_entry()`, `ldap_next_entry()`, `ldap_first_reference()`, `ldap_next_reference()`.

Parsing Search Results

Use the functions in this section to parse the entries and references returned by `ldap_search` functions. These results are returned in an opaque structure that may be accessed by calling the routines described in this section. Routines are provided to step through the entries and references returned, step through the attributes of an entry, retrieve the name of an entry, and retrieve the values associated with a given attribute in an entry.

ldap_first_entry, ldap_next_entry, ldap_first_reference, ldap_next_reference, ldap_count_entries, and ldap_count_references

The `ldap_first_entry()` and `ldap_next_entry()` routines are used to step through and retrieve the list of entries from a search result chain. The `ldap_first_reference()` and `ldap_next_reference()` routines are used to step through and retrieve the list of continuation references from a search result chain. `ldap_count_entries()` is used to count the number of entries returned. `ldap_count_references()` is used to count the number of references returned.

```
LDAPMessage *ldap_first_entry( LDAP *ld, LDAPMessage *res );
LDAPMessage *ldap_next_entry( LDAP *ld, LDAPMessage *entry );
LDAPMessage *ldap_first_reference( LDAP *ld, LDAPMessage *res );
LDAPMessage *ldap_next_reference( LDAP *ld, LDAPMessage *ref );
int ldap_count_entries( LDAP *ld, LDAPMessage *res );
int ldap_count_references( LDAP *ld, LDAPMessage *res );
```

Parameters

Table 8–23 lists and describes the parameters or retrieving entries and continuation references from a search result chain, and for counting entries returned.

Table 8–23 Parameters for Retrieving Entries and Continuation References from a Search Result Chain, and for Counting Entries Returned

Parameter	Description
`ld`	The session handle.
`res`	The search result, as obtained by a call to one of the synchronous search routines or `ldap_result()`.
`entry`	The entry returned by a previous call to `ldap_first_entry()` or `ldap_next_entry()`.
`ref`	The reference returned by a previous call to `ldap_first_reference()` or `ldap_next_reference()`.

Usage Notes

`ldap_first_entry()`, `ldap_next_entry()`, `ldap_first_reference()`, and `ldap_next_reference()` all return `NULL` when no more entries or references exist

in the result set to be returned. NULL is also returned if an error occurs while stepping through the entries or references, in which case the error parameters in the session handle ld will be set to indicate the error.

ldap_count_entries() returns the number of entries contained in a chain of entries; if an error occurs such as the res parameter being invalid, -1 is returned. The ldap_count_entries() call can also be used to count the number of entries that remain in a chain if called with a message, entry or reference returned by ldap_first_message(), ldap_next_message(), ldap_first_entry(), ldap_next_entry(), ldap_first_reference(), ldap_next_reference().

ldap_count_references() returns the number of references contained in a chain of search results; if an error occurs such as the res parameter being invalid, -1 is returned. The ldap_count_references() call can also be used to count the number of references that remain in a chain.

ldap_first_attribute and ldap_next_attribute

Use the functions in this section to step through the list of attribute types returned with an entry.

Syntax

```
char *ldap_first_attribute
(
LDAP            *ld,
LDAPMessage     *entry,
BerElement      **ptr
);

char *ldap_next_attribute
(
LDAP            *ld,
LDAPMessage     *entry,
BerElement      *ptr
);
void ldap_memfree( char *mem );
```

Parameters

Table 8–24 lists and describes the parameters for stepping through attribute types returned with an entry.

Table 8–24 Parameters for Stepping Through Attribute Types Returned with an Entry

Parameter	Description
ld	The session handle.
entry	The entry whose attributes are to be stepped through, as returned by ldap_first_entry() or ldap_next_entry().
ptr	In ldap_first_attribute(), the address of a pointer used internally to keep track of the current position in the entry. In ldap_next_attribute(), the pointer returned by a previous call to ldap_first_attribute(). The BerElement type itself is an opaque structure.
mem	A pointer to memory allocated by the LDAP library, such as the attribute type names returned by ldap_first_attribute() and ldap_next_attribute, or the DN returned by ldap_get_dn(). If mem is NULL, the ldap_memfree() call does nothing.

Usage Notes

`ldap_first_attribute()` and `ldap_next_attribute()` returns `NULL` when the end of the attributes is reached, or if there is an error. In the latter case, the error parameters in the session handle `ld` are set to indicate the error.

Both routines return a pointer to an allocated buffer containing the current attribute name. This should be freed when no longer in use by calling `ldap_memfree()`.

`ldap_first_attribute()` will allocate and return in `ptr` a pointer to a `BerElement` used to keep track of the current position. This pointer may be passed in subsequent calls to `ldap_next_attribute()` to step through the entry's attributes. After a set of calls to `ldap_first_attribute()` and `ldap_next_attribute()`, if `ptr` is non-null, it should be freed by calling `ber_free(ptr, 0)`. Note that it is very important to pass the second parameter as `0` (zero) in this call, since the buffer associated with the `BerElement` does not point to separately allocated memory.

The attribute type names returned are suitable for passing in a call to `ldap_get_values()` and friends to retrieve the associated values.

ldap_get_values, ldap_get_values_len, ldap_count_values, ldap_count_values_len, ldap_value_free, and ldap_value_free_len

`ldap_get_values()` and `ldap_get_values_len()` are used to retrieve the values of a given attribute from an entry. `ldap_count_values()` and `ldap_count_values_len()` are used to count the returned values.

`ldap_value_free()` and `ldap_value_free_len()` are used to free the values.

Syntax

```
char **ldap_get_values
(
LDAP            *ld,
LDAPMessage     *entry,
const char      *attr
);

struct berval **ldap_get_values_len
(
LDAP            *ld,
LDAPMessage     *entry,
const char      *attr
);

int ldap_count_values( char **vals );
int ldap_count_values_len( struct berval **vals );
void ldap_value_free( char **vals );
void ldap_value_free_len( struct berval **vals );
```

Parameters

Table 8–25 lists and describes the parameters for retrieving and counting attribute values.

Table 8–25 Parameters for Retrieving and Counting Attribute Values

Parameter	Description
`ld`	The session handle.
`entry`	The entry from which to retrieve values, as returned by `ldap_first_entry()` or `ldap_next_entry()`.

Table 8–25 (Cont.) Parameters for Retrieving and Counting Attribute Values

Parameter	Description
attr	The attribute whose values are to be retrieved, as returned by ldap_first_attribute() or ldap_next_attribute(), or a caller-supplied string (for example, "mail").
vals	The values returned by a previous call to ldap_get_values() or ldap_get_values_len().

Usage Notes

Two forms of the various calls are provided. The first form is only suitable for use with non-binary character string data. The second _len form is used with any kind of data.

ldap_get_values() and ldap_get_values_len() return NULL if no values are found for attr or if an error occurs.

ldap_count_values() and ldap_count_values_len() return -1 if an error occurs such as the vals parameter being invalid.

If a NULL vals parameter is passed to ldap_value_free() or ldap_value_free_len(), nothing is done.

Note that the values returned are dynamically allocated and should be freed by calling either ldap_value_free() or ldap_value_free_len() when no longer in use.

ldap_get_dn, ldap_explode_dn, ldap_explode_rdn, and ldap_dn2ufn

ldap_get_dn() is used to retrieve the name of an entry. ldap_explode_dn() and ldap_explode_rdn() are used to break up a name into its component parts. ldap_dn2ufn() is used to convert the name into a more user friendly format.

Syntax

```
char *ldap_get_dn( LDAP *ld, LDAPMessage *entry );
char **ldap_explode_dn( const char *dn, int notypes );
char **ldap_explode_rdn( const char *rdn, int notypes );
char *ldap_dn2ufn( const char *dn );
```

Parameters

Table 8–26 lists and describes the parameters for retrieving, exploding, and converting entry names.

Table 8–26 Parameters for Retrieving, Exploding, and Converting Entry Names

Parameter	Description
ld	The session handle.
entry	The entry whose name is to be retrieved, as returned by ldap_first_entry() or ldap_next_entry().
dn	The DN to explode, such as returned by ldap_get_dn().
rdn	The RDN to explode, such as returned in the components of the array returned by ldap_explode_dn().
notypes	A Boolean parameter, if nonzero indicating that the DN or RDN components are to have their type information stripped off: cn=Babs would become Babs.

Usage Notes

`ldap_get_dn()` returns `NULL` if a DN parsing error occurs. The function sets error parameters in the session handle `ld` to indicate the error. It returns a pointer to newly allocated space that the caller should free by calling `ldap_memfree()` when it is no longer in use.

`ldap_explode_dn()` returns a `NULL`-terminated `char *` array containing the RDN components of the DN supplied, with or without types as indicated by the `notypes` parameter. The components are returned in the order they appear in the DN. The array returned should be freed when it is no longer in use by calling `ldap_value_free()`.

`ldap_explode_rdn()` returns a `NULL`-terminated `char *` array containing the components of the RDN supplied, with or without types as indicated by the `notypes` parameter. The components are returned in the order they appear in the rdn. The array returned should be freed when it is no longer in use by calling `ldap_value_free()`.

`ldap_dn2ufn()` converts the DN into a user friendly format. The UFN returned is newly allocated space that should be freed by a call to `ldap_memfree()` when no longer in use.

ldap_get_entry_controls

`ldap_get_entry_controls()` is used to extract LDAP controls from an entry.

Syntax

```
int ldap_get_entry_controls
(
LDAP            *ld,
LDAPMessage     *entry,
LDAPControl     ***serverctrlsp
);
```

Parameters

Table 8–27 lists and describes the parameters for extracting LDAP control from an entry.

Table 8–27 Parameters for Extracting LDAP Controls from an Entry

Parameters	Description
`ld`	The session handle.
`entry`	The entry to extract controls from, as returned by `ldap_first_entry()` or `ldap_next_entry()`.
`serverctrlsp`	This result parameter will be filled in with an allocated array of controls copied out of entry. The control array should be freed by calling `ldap_controls_free()`. If `serverctrlsp` is `NULL`, no controls are returned.

Usage Notes

`ldap_get_entry_controls()` returns an LDAP error code that indicates whether the reference could be successfully parsed (`LDAP_SUCCESS` if all goes well).

ldap_parse_reference

Use `ldap_parse_reference()` to extract referrals and controls from a `SearchResultReference` message.

Syntax

```
int ldap_parse_reference
(
LDAP            *ld,
LDAPMessage     *ref,
char            ***referralsp,
LDAPControl     ***serverctrlsp,
int             freeit
);
```

Parameters

Table 8–28 lists and describes parameters for extracting referrals and controls from a `SearchResultReference` message.

Table 8–28 Parameters for Extracting Referrals and Controls from a SearchResultReference Message

Parameter	Description
`ld`	The session handle.
`ref`	The reference to parse, as returned by `ldap_result()`, `ldap_first_reference()`, or `ldap_next_reference()`.
`referralsp`	This result parameter will be filled in with an allocated array of character strings. The elements of the array are the referrals (typically LDAP URLs) contained in ref. The array should be freed when no longer in used by calling `ldap_value_free()`. If `referralsp` is `NULL`, the referral URLs are not returned.
`serverctrlsp`	This result parameter will be filled in with an allocated array of controls copied out of `ref`. The control array shouldbe freed by calling `ldap_controls_free()`. If `serverctrlsp` is `NULL`, no controls are returned.
`freeit`	A Boolean that determines whether the `ref` parameter is disposed of or not. Pass any nonzero value to have this routine free `ref` after extracting the requested information. This is provided as a convenience. You can also use `ldap_msgfree()` to free the result later.

Usage Notes

`ldap_parse_reference()` returns an LDAP error code that indicates whether the reference could be successfully parsed (`LDAP_SUCCESS` if all goes well).

Sample C API Usage

The following examples show how to use the C API both with and without SSL and for SASL authentication. More complete examples are given in RFC 1823. The sample code for the command-line tool to perform an LDAP search also demonstrates use of the API in both the SSL and the non-SSL mode.

This section contains these topics:

- C API Usage with SSL
- C API Usage Without SSL
- C API Usage for SASL-Based DIGEST-MD5 Authentication

C API Usage with SSL

```
#include <stdio.h>
#include <ldap.h>

main()
{
LDAP          *ld;
int           ret = 0;
….
/* open a connection */
if ((ld = ldap_open("MyHost", 636)) == NULL)
     exit( 1 );

/* SSL initialization */
ret = ldap_init_SSL(&ld->ld_sb, "file:/sslwallet", "welcome",
                                          GSLC_SSL_ONEWAY_AUTH );
if(ret != 0)
{
printf(" %s \n", ldap_err2string(ret));
exit(1);
}

/* authenticate as nobody */
if ( ldap_bind_s( ld, NULL, NULL ) != LDAP_SUCCESS ) {
     ldap_perror( ld, "ldap_bind_s" );
     exit( 1 );
}

.
.
.
}
```

Because the user is making the `ldap_init_SSL` call, the client/server communication in the previous example is secured by using SSL.

C API Usage Without SSL

```
#include <stdio.h>
#include <ldap.h>

main()
{
LDAP          *ld;
int           ret = 0;
.
.
.
/* open a connection */
if ( (ld = ldap_open( "MyHost", LDAP_PORT
 )) == NULL )
     exit( 1 );

/* authenticate as nobody */
if ( ldap_bind_s( ld, NULL, NULL ) != LDAP_SUCCESS ) {
     ldap_perror( ld, "ldap_bind_s" );
      exit( 1 );
}
```

```
.
.
.
}
```

In the previous example, the user is not making the `ldap_init_SSL` call, and the client-to-server communication is therefore not secure.

C API Usage for SASL-Based DIGEST-MD5 Authentication

This sample program illustrates the usage of LDAP SASL C-API for SASL-based DIGEST-MD5 authentication to a directory server.

```
/*
   EXPORT FUNCTION(S)
     NONE

   INTERNAL FUNCTION(S)
     NONE

   STATIC FUNCTION(S)
     NONE

   NOTES
     Usage:
      saslbind -h ldap_host -p ldap_port -D authentication_identity_dn \
               -w <password >
      options
       -h     LDAP host
       -p     LDAP port
       -D     DN of the identity for authentication
       -p     Password

       Default SASL authentication parameters used by the demo program
       SASL Security Property :    Currenty only "auth" security property
                                   is supported by the C-API. This demo
                                   program uses this security property.
       SASL Mechanism         :    Supported mechanisms by OID
                                   "DIGEST-MD5" - This demo program
                                                  illustrates it's usage.
                                   "EXTERNAL" - SSL authentication is used.
                                                (This demo program does
                                                 not illustrate it's usage.)
       Authorization identity :    This demo program does not use any
                                    authorization identity.

   MODIFIED   (MM/DD/YY)
   ******      06/12/03 - Creation

*/

/*---------------------------------------------------------------------------
                     PRIVATE TYPES AND CONSTANTS
  ---------------------------------------------------------------------------*/

/*---------------------------------------------------------------------------
                     STATIC FUNCTION DECLARATIONS
  ---------------------------------------------------------------------------*/

#include <stdio.h>
```

```
#include <stdlib.h>
#include <ldap.h>


static int ldap_version = LDAP_VERSION3;

main (int argc, char **argv)
{
  LDAP*           ld;
  extern char*    optarg;
  char*           ldap_host = NULL;
  char*           ldap_bind_dn = NULL;
  char*           ldap_bind_pw = NULL;
  int             authmethod = 0;
  char            ldap_local_host[256] = "localhost";
  int             ldap_port = 389;
  char*           authcid = (char *)NULL;
  char*           mech = "DIGEST-MD5"; /* SASL mechanism */
  char*           authzid = (char *)NULL;
  char*           sasl_secprops = "auth";
  char*           realm = (char *)NULL;
  int             status = LDAP_SUCCESS;
  OraLdapHandle   sasl_cred = (OraLdapHandle )NULL;
  OraLdapClientCtx *cctx = (OraLdapClientCtx *)NULL;
  int              i = 0;


    while (( i = getopt( argc, argv,
    "D:h:p:w:E:P:U:V:W:O:R:X:Y:Z"
    )) != EOF ) {
switch( i ) {

case 'h':/* ldap host */
    ldap_host = (char *)strdup( optarg );
    break;
case 'D':/* bind DN */
    authcid = (char *)strdup( optarg );
    break;

case 'p':/* ldap port */
    ldap_port = atoi( optarg );
    break;
case 'w':/* Password */
    ldap_bind_pw = (char *)strdup( optarg );
    break;

    default:
    printf("Invalid Arguments passed\n" );
}
    }


  /* Get the connection to the LDAP server */
  if (ldap_host == NULL)
    ldap_host = ldap_local_host;

  if ((ld = ldap_open (ldap_host, ldap_port)) == NULL)
  {
    ldap_perror (ld, "ldap_init");
    exit (1);
```

```
  }

  /* Create the client context needed by LDAP C-API Oracle Extension functions*/
  status = ora_ldap_init_clientctx(&cctx);

  if(LDAP_SUCCESS != status) {
     printf("Failed during creation of client context \n");
     exit(1);
  }

  /* Create SASL credentials */
  sasl_cred = ora_ldap_create_cred_hdl(cctx, ORA_LDAP_CRED_HANDLE_SASL_MD5);

  ora_ldap_set_cred_props(cctx, sasl_cred, ORA_LDAP_CRED_SASL_REALM, (void
*)realm);
  ora_ldap_set_cred_props(cctx, sasl_cred, ORA_LDAP_CRED_SASL_AUTH_PASSWORD, (void
*)ldap_bind_pw);
  ora_ldap_set_cred_props(cctx, sasl_cred, ORA_LDAP_CRED_SASL_AUTHORIZATION_
ID,(void *)authzid);
  ora_ldap_set_cred_props(cctx, sasl_cred, ORA_LDAP_CRED_SASL_SECURITY_PROPERTIES,
(void *)sasl_secprops);

  /* If connecting to the directory using SASL DIGEST-MD5, the Authentication ID
     has to be normalized before it's sent to the server,
     the LDAP C-API does this normalization based on the following flag set in
     SASL credential properties */
  ora_ldap_set_cred_props(cctx, sasl_cred, ORA_LDAP_CRED_SASL_NORM_AUTHDN, (void
*)NULL);

  /* SASL Authetication to LDAP Server */
  status = (int)ora_ldap_init_SASL(cctx, ld, (char *)authcid, (char *)ORA_LDAP_
SASL_MECH_DIGEST_MD5,
     sasl_cred, NULL, NULL);

  if(LDAP_SUCCESS == status) {
     printf("SASL bind successful \n" );
  }else {
     printf("SASL bind failed with status :  %d\n", status);
  }

  /* Free SASL Credentials */
  ora_ldap_free_cred_hdl(cctx, sasl_cred);

  status = ora_ldap_free_clientctx(cctx);

  /* Unbind from LDAP server */
  ldap_unbind (ld);

  return (0);
}


/* end of file saslbind.c */
```

Required Header Files and Libraries for the C API

To build applications with the C API, you need to:

- Include the header file located at *ORACLE_HOME*/ldap/public/ldap.h.

- Dynamically link to the library located at
  - `$ORACLE_HOME/lib/libclntsh.so.10.1` on UNIX operating systems
  - `%ORACLE_HOME%\bin\oraldapclnt10.dll` on Windows operating systems

Dependencies and Limitations of the C API

This API can work against any release of Oracle Internet Directory. It requires either an Oracle environment or, at minimum, globalization support and other core libraries.

To use the different authentication modes in SSL, the directory server requires corresponding configuration settings.

> **See Also:** *Oracle Internet Directory Administrator's Guide* for details about how to set the directory server in various SSL authentication modes

Oracle Wallet Manager is required for creating wallets if you are using the C API in SSL mode.

TCP/IP Socket Library is required.

The following Oracle libraries are required:

- Oracle SSL-related libraries
- Oracle system libraries

Sample libraries are included in the release for the sample command line tool. You should replace these libraries with your own versions of the libraries.

The product supports only those authentication mechanisms described in LDAP SDK specifications (RFC 1823).

9

DBMS_LDAP PL/SQL Reference

DBMS_LDAP contains the functions and procedures that enable PL/SQL programmers to access data from LDAP servers. This chapter examines all of the API functions in detail.

The chapter contains these topics:

- Summary of Subprograms
- Exception Summary
- Data Type Summary
- Subprograms

> **Note:** Sample code for the DBMS_LDAP package is available at this URL:
>
> http://www.oracle.com/technology/sample_code/id_mgmt

Summary of Subprograms

Table 9–1 DBMS_LDAP API Subprograms

Function or Procedure	Description
FUNCTION init	init() initializes a session with an LDAP server. This actually establishes a connection with the LDAP server.
FUNCTION simple_bind_s	The function simple_bind_s() can be used to perform simple user name and password authentication to the directory server.
FUNCTION bind_s	The function bind_s() can be used to perform complex authentication to the directory server.
FUNCTION unbind_s	The function unbind_s() is used for closing an active LDAP session.
FUNCTION compare_s	The function compare_s() can be used to test if a particular attribute in a particular entry has a particular value.
FUNCTION search_s	The function search_s() performs a synchronous search in the LDAP server. It returns control to the PL/SQL environment only after all of the search results have been sent by the server or if the search request is 'timed-out by the server.

Table 9–1 (Cont.) DBMS_LDAP API Subprograms

Function or Procedure	Description
FUNCTION search_st	The function `search_st()` performs a synchronous search in the LDAP server with a client side time out. It returns control to the PL/SQL environment only after all of the search results have been sent by the server or if the search request is 'timed-out' by the client or the server.
FUNCTION first_entry	The function first_entry is used to retrieve the first entry in the result set returned by either `search_s()` or search_st.
FUNCTION next_entry	The function `next_entry()` is used to iterate to the next entry in the result set of a search operation.
FUNCTION count_entries	This function is used to count the number of entries in the result set. It can also be used to count the number of entries remaining during a traversal of the result set using a combination of the functions `first_entry()` and `next_entry`.
FUNCTION first_attribute	The function `first_attribute()` fetches the first attribute of a given entry in the result set.
FUNCTION next_attribute	The function `next_attribute()`fetches the next attribute of a given entry in the result set.
FUNCTION get_dn	The function `get_dn()` retrieves the X.500 distinguished name of a given entry in the result set.
FUNCTION get_values	The function `get_values()`can be used to retrieve all of the values associated with a given attribute in a given entry.
FUNCTION get_values_len	The function `get_values_len()` can be used to retrieve values of attributes that have a 'Binary' syntax.
FUNCTION delete_s	This function can be used to remove a leaf entry in the LDAP Directory Information Tree.
FUNCTION modrdn2_s	The function `modrdn2_s()` can be used to rename the relative distinguished name of an entry.
FUNCTION err2string	The function `err2string()` can be used to convert an LDAP error code to a string in the local language in which the API is operating.
FUNCTION create_mod_array	The function `create_mod_array()` allocates memory for array modification entries that will be applied to an entry using the `modify_s()` functions.
PROCEDURE populate_mod_array (String Version)	Populates one set of attribute information for add or modify operations. This procedure call has to happen after `DBMS_LDAP.create_mod_array()` is called.
PROCEDURE populate_mod_array (Binary Version)	Populates one set of attribute information for add or modify operations. This procedure call has to occur after `DBMS_LDAP.create_mod_array()` is called.
PROCEDURE populate_mod_array (Binary Version. Uses BLOB Data Type)	Populates one set of attribute information for add or modify operations. This procedure call has to happen after `DBMS_LDAP.create_mod_array()` is called.
FUNCTION get_values_blob	The function `get_values_blob()` can be used to retrieve larger values of attributes that have a binary syntax.
FUNCTION count_values_blob	Counts the number of values returned by `DBMS_LDAP.get_values_blob()`.

Table 9–1 (Cont.) DBMS_LDAP API Subprograms

Function or Procedure	Description
FUNCTION value_free_blob	Frees the memory associated with the `BLOB_COLLECTION` returned by `DBMS_LDAP.get_values_blob()`.
FUNCTION modify_s	Performs a synchronous modification of an existing LDAP directory entry. Before calling `add_s`, you must call `DBMS_LDAP.creat_mod_array()` and `DBMS_LDAP.populate_mod_array()`.
FUNCTION add_s	Adds a new entry to the LDAP directory synchronously. Before calling `add_s`, you must call `DBMS_LDAP.creat_mod_array()` and `DBMS_LDAP.populate_mod_array()`.
PROCEDURE free_mod_array	Frees the memory allocated by `DBMS_LDAP.create_mod_array()`.
FUNCTION count_values	Counts the number of values returned by `DBMS_LDAP.get_values()`.
FUNCTION count_values_len	Counts the number of values returned by `DBMS_LDAP.get_values_len ()`.
FUNCTION rename_s	Renames an LDAP entry synchronously.
FUNCTION explode_dn	Breaks a DN up into its components.
FUNCTION open_ssl	Establishes an SSL (Secure Sockets Layer) connection over an existing LDAP connection.
FUNCTION msgfree	This function frees the chain of messages associated with the message handle returned by synchronous search functions.
FUNCTION ber_free	This function frees the memory associated with a handle to `BER_ELEMENT`.
FUNCTION nls_convert_to_utf8	The `nls_convert_to_utf8` function converts the input string containing database character set data to UTF8 character set data and returns it.
FUNCTION nls_convert_from_utf8	The `nls_convert_from_utf8` function converts the input string containing UTF8 character set data to database character set data and returns it.
FUNCTION nls_get_dbcharset_name	The `nls_get_dbcharset_name` function returns a string containing the database character set name.

See Also:

- "Searching the Directory" in Chapter 3 for more about `DBMS_LDAP.search_s()` and `DBMS_LDAP.search_st()`
- "Terminating the Session by Using DBMS_LDAP" in Chapter 3 for more about `DBMS_LDAP.unbind_s()`

Exception Summary

`DBMS_LDAP` can generate the exceptions described in Table 9–2 on page 9-4.

Table 9–2 DBMS_LDAP Exception Summary

Exception Name	Oracle Error Number	Cause of Exception
general_error	31202	Raised anytime an error is encountered that does not have a specific PL/SQL exception associated with it. The error string contains the description of the problem in the user's language.
init_failed	31203	Raised by DBMS_LDAP.init() if there are problems.
invalid_session	31204	Raised by all functions and procedures in the DBMS_LDAP package if they are passed an invalid session handle.
invalid_auth_method	31205	Raised by DBMS_LDAP.bind_s()if the authentication method requested is not supported.
invalid_search_scope	31206	Raised by all search functions if the scope of the search is invalid.
invalid_search_time_val	31207	Raised by DBMS_LDAP.search_st()if it is given an invalid value for a time limit.
invalid_message	31208	Raised by all functions that iterate through a result-set for getting entries from a search operation if the message handle given to them is invalid.
count_entry_error	31209	Raised by DBMS_LDAP.count_entries if it cannot count the entries in a given result set.
get_dn_error	31210	Raised by DBMS_LDAP.get_dn if the DN of the entry it is retrieving is NULL.
invalid_entry_dn	31211	Raised by all functions that modify, add, or rename an entry if they are presented with an invalid entry DN.
invalid_mod_array	31212	Raised by all functions that take a modification array as an argument if they are given an invalid modification array.
invalid_mod_option	31213	Raised by DBMS_LDAP.populate_mod_array if the modification option given is anything other than MOD_ADD, MOD_DELETE or MOD_REPLACE.
invalid_mod_type	31214	Raised by DBMS_LDAP.populate_mod_array if the attribute type that is being modified is NULL.
invalid_mod_value	31215	Raised by DBMS_LDAP.populate_mod_array if the modification value parameter for a given attribute is NULL.
invalid_rdn	31216	Raised by all functions and procedures that expect a valid RDN and are provided with an invalid one.
invalid_newparent	31217	Raised by DBMS_LDAP.rename_s if the new parent of an entry being renamed is NULL.
invalid_deleteoldrdn	31218	Raised by DBMS_LDAP.rename_s if the deleteoldrdn parameter is invalid.
invalid_notypes	31219	Raised by DBMS_LDAP.explode_dn if the notypes parameter is invalid.

Table 9–2 (Cont.) DBMS_LDAP Exception Summary

Exception Name	Oracle Error Number	Cause of Exception
invalid_ssl_wallet_loc	31220	Raised by DBMS_LDAP.open_ssl if the wallet location is NULL but the SSL authentication mode requires a valid wallet.
invalid_ssl_wallet_password	31221	Raised by DBMS_LDAP.open_ssl if the wallet password given is NULL.
invalid_ssl_auth_mode	31222	Raised by DBMS_LDAP.open_ssl if the SSL authentication mode is not 1, 2 or 3.

Data Type Summary

The DBMS_LDAP package uses the data types described in Table 9–3.

Table 9–3 DBMS_LDAP Data Type Summary

Data-Type	Purpose
SESSION	Used to hold the handle of the LDAP session. Nearly all of the functions in the API require a valid LDAP session to work.
MESSAGE	Used to hold a handle to the message retrieved from the result set. This is used by all functions that work with entry attributes and values.
MOD_ARRAY	Used to hold a handle to the array of modifications being passed to either modify_s() or add_s().
TIMEVAL	Used to pass time limit information to the LDAP API functions that require a time limit.
BER_ELEMENT	Used to hold a handle to a BER structure used for decoding incoming messages.
STRING_COLLECTION	Used to hold a list of VARCHAR2 strings that can be passed on to the LDAP server.
BINVAL_COLLECTION	Used to hold a list of RAW data, which represent binary data.
BERVAL_COLLECTION	Used to hold a list of BERVAL values that are used for populating a modification array.
BLOB_COLLECTION	Used to hold a list of BLOB data, which represent binary data.

Subprograms

This section takes a closer look at each of the DBMS_LDAP subprograms.

FUNCTION init

init() initializes a session with an LDAP server. This actually establishes a connection with the LDAP server.

Syntax

```
FUNCTION init
(
hostname IN VARCHAR2,
portnum  IN PLS_INTEGER
)
RETURN SESSION;
```

Parameters

Table 9–4 INIT Function Parameters

Parameter	Description
hostname	Contains a space-separated list of host names or dotted strings representing the IP address of hosts running an LDAP server to connect to. Each host name in the list may include a port number, which is separated from the host by a colon. The hosts are tried in the order listed, stopping with the first one to which a successful connection is made.
portnum	Contains the TCP port number to connect to. If the port number is included with the host name, this parameter is ignored. If the parameter is not specified, and the host name does not contain the port number, a default port number of 389 is assumed.

Return Values

Table 9–5 INIT Function Return Values

Value	Description
SESSION	A handle to an LDAP session that can be used for further calls to the API.

Exceptions

Table 9–6 INIT Function Exceptions

Exception	Description
init_failed	Raised when there is a problem contacting the LDAP server.
general_error	For all other errors. The error string associated with the exception describes the error in detail.

Usage Notes

DBMS_LDAP.init() is the first function that should be called because it establishes a session with the LDAP server. Function DBMS_LDAP.init() returns a session handle, a pointer to an opaque structure that must be passed to subsequent calls pertaining to the session. This routine will return NULL and raise the INIT_FAILED exception if the session cannot be initialized. After init() has been called, the connection has to be authenticated using DBMS_LDAP.bind_s or DBMS_LDAP.simple_bind_s().

See Also

DBMS_LDAP.simple_bind_s(), DBMS_LDAP.bind_s().

FUNCTION simple_bind_s

The function `simple_bind_s` can be used to perform simple user name and password authentication to the directory server.

Syntax

```
FUNCTION simple_bind_s
(
ld     IN SESSION,
dn     IN VARCHAR2,
passwd IN VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters

Table 9–7 SIMPLE_BIND_S Function Parameters

Parameter	Description
`ld`	A valid LDAP session handle.
`dn`	The Distinguished Name of the User that we are trying to login as.
`passwd`	A text string containing the password.

Return Values

Table 9–8 SIMPLE_BIND_S Function Return Values

Value	Description
`PLS_INTEGER`	`DBMS_LDAP.SUCCESS` on a successful completion. If there was a problem, one of the following exceptions will be raised.

Exceptions

Table 9–9 SIMPLE_BIND_S Function Exceptions

Exception	Description
`invalid_session`	Raised if the session handle `ld` is invalid.
`general_error`	For all other errors. The error string associated with this exception will explain the error in detail.

Usage Notes

`DBMS_LDAP.simple_bind_s()` can be used to authenticate a user whose directory distinguished name and directory password are known. It can be called only after a valid LDAP session handle is obtained from a call to `DBMS_LDAP.init()`.

FUNCTION bind_s

The function `bind_s` can be used to perform complex authentication to the directory server.

Syntax

```
FUNCTION  bind_s
(
```

```
ld     IN SESSION,
dn     IN VARCHAR2,
cred IN VARCHAR2,
meth IN PLS_INTEGER
)
RETURN PLS_INTEGER;
```

Parameters

Table 9–10 BIND_S Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
dn	The distinguished name of the user.
cred	A text string containing the credentials used for authentication.
meth	The authentication method.

Return Values

Table 9–11 BIND_S Function Return Values

Value	Description
PLS_INTEGER	DBMS_LDAP.SUCCESS upon successful completion. One of the following exceptions is raised if there is a problem.

Exceptions

Table 9–12 BIND_S Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_auth_method	Raised if the authentication method requested is not supported.
general_error	For all other errors. The error string associated with this exception will explain the error in detail.

Usage Notes

DBMS_LDAP.bind_s() can be used to authenticate a user. It can be called only after a valid LDAP session handle is obtained from a call to DBMS_LDAP.init().

See Also

DBMS_LDAP.init(), DBMS_LDAP.simple_bind_s().

FUNCTION unbind_s

The function unbind_s is used for closing an active LDAP session.

Syntax

```
FUNCTION unbind_s
(
ld IN OUT SESSION
)
RETURN PLS_INTEGER;
```

Parameters

Table 9–13 UNBIND_S Function Parameters

Parameter	Description
ld	A valid LDAP session handle.

Return Values

Table 9–14 UNBIND_S Function Return Values

Value	Description
PLS_INTEGER	DBMS_LDAP.SUCCESS on proper completion. One of the following exceptions is raised otherwise.

Exceptions

Table 9–15 UNBIND_S Function Exceptions

Exception	Description
invalid_session	Raised if the sessions handle ld is invalid.
general_error	For all other errors. The error string associated with this exception will explain the error in detail.

Usage Notes

The unbind_s() function sends an unbind request to the server, closes all open connections associated with the LDAP session, and disposes of all resources associated with the session handle before returning. After a call to this function, the session handle ld is invalid.

See Also

DBMS_LDAP.bind_s(), DBMS_LDAP.simple_bind_s().

FUNCTION compare_s

The function compare_s can be used to test if a particular attribute in a particular entry has a particular value.

Syntax

```
FUNCTION compare_s
(
ld    IN SESSION,
dn    IN VARCHAR2,
attr  IN VARCHAR2,
value IN VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters

Table 9–16 COMPARE_S Function Parameters

Parameter	Description
ld	A valid LDAP session handle.

Table 9–16 (Cont.) COMPARE_S Function Parameters

Parameter	Description
dn	The name of the entry to compare against.
attr	The attribute to compare against.
value	A string attribute value to compare against.

Return Values

Table 9–17 COMPARE_S Function Return Values

Value	Description
PLS_INTEGER	COMPARE_TRUE if the given attribute has a matching value. COMPARE_FALSE if the given attribute does not have a matching value.

Exceptions

Table 9–18 COMPARE_S Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
general_error	For all other errors. The error string associated with this exception will explain the error in detail.

Usage Notes

The function compare_s can be used to assert that an attribute in the directory has a certain value. This operation can be performed only on attributes whose syntax enables them to be compared. The compare_s function can be called only after a valid LDAP session handle has been obtained from the init() function and authenticated by the bind_s() or simple_bind_s() functions.

See Also

DBMS_LDAP.bind_s()

FUNCTION search_s

The function search_s performs a synchronous search in the directory. It returns control to the PL/SQL environment only after all of the search results have been sent by the server or if the search request is timed out by the server.

Syntax

```
FUNCTION search_s
(
ld       IN  SESSION,
base     IN  VARCHAR2,
scope    IN  PLS_INTEGER,
filter   IN  VARCHAR2,
attrs    IN  STRING_COLLECTION,
attronly IN  PLS_INTEGER,
res      OUT MESSAGE
)
RETURN PLS_INTEGER;
```

Parameters

Table 9–19 SEARCH_S Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
base	The DN of the entry at which to start the search.
scope	One of SCOPE_BASE (0x00), SCOPE_ONELEVEL (0x01), or SCOPE_SUBTREE (0x02), indicating the scope of the search.
filter	A character string representing the search filter. The value NULL can be passed to indicate that the filter "(objectclass=*)", which matches all entries, is to be used.
attrs	A collection of strings indicating which attributes to return for each matching entry. Passing NULL for this parameter causes all available user attributes to be retrieved. The special constant string NO_ATTRS ("1.1") may be used as the only string in the array to indicate that no attribute types are to be returned by the server. The special constant string ALL_USER_ATTRS ("*") can be used in the attrs array along with the names of some operational attributes to indicate that all user attributes plus the listed operational attributes are to be returned.
attrsonly	A boolean value that must be zero if both attribute types and values are to be returned, and nonzero if only types are wanted.
res	This is a result parameter that contains the results of the search upon completion of the call. If no results are returned, *res is set to NULL.

Return Values

Table 9–20 SEARCH_S Function Return Value

Value	Description
PLS_INTEGER	DBMS_LDAP.SUCCESS if the search operation succeeded. An exception is raised in all other cases.
res	If the search succeeded and there are entries, this parameter is set to a non-null value which can be used to iterate through the result set.

Exceptions

Table 9–21 SEARCH_S Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_search_scope	Raised if the search scope is not one of SCOPE_BASE, SCOPE_ONELEVEL, or SCOPE_SUBTREE.
general_error	For all other errors. The error string associated with this exception will explain the error in detail.

Usage Notes

The function search_s() issues a search operation and does not return control to the user environment until all of the results have been returned from the server. Entries returned from the search, if any, are contained in the res parameter. This parameter is opaque to the caller. Entries, attributes, and values can be extracted by calling the parsing routines described in this chapter.

See Also

DBMS_LDAP.search_st(), DBMS_LDAP.first_entry(), DBMS_LDAP.next_entry.

FUNCTION search_st

The function search_st() performs a synchronous search in the LDAP server with a client-side time out. It returns control to the PL/SQL environment only after all of the search results have been sent by the server or if the search request is timed out by the client or the server.

Syntax

```
FUNCTION search_st
(
ld       IN  SESSION,
base     IN  VARCHAR2,
scope    IN  PLS_INTEGER,
filter   IN  VARCHAR2,
attrs    IN  STRING_COLLECTION,
attronly IN  PLS_INTEGER,
tv       IN  TIMEVAL,
res      OUT MESSAGE
)
RETURN PLS_INTEGER;
```

Parameters

Table 9–22 SEARCH_ST Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
base	The DN of the entry at which to start the search.
scope	One of SCOPE_BASE (0x00), SCOPE_ONELEVEL (0x01), or SCOPE_SUBTREE (0x02), indicating the scope of the search.
filter	A character string representing the search filter. The value NULL can be passed to indicate that the filter "(objectclass=*)", which matches all entries, is to be used.
attrs	A collection of strings indicating which attributes to return for each matching entry. Passing NULL for this parameter causes all available user attributes to be retrieved. The special constant string NO_ATTRS ("1.1") may be used as the only string in the array to indicate that no attribute types are to be returned by the server. The special constant string ALL_USER_ATTRS ("*") can be used in the attrs array along with the names of some operational attributes to indicate that all user attributes plus the listed operational attributes are to be returned.
attrsonly	A boolean value that must be zero if both attribute types and values are to be returned, and nonzero if only types are wanted.
tv	The time out value, expressed in seconds and microseconds, that should be used for this search.
res	This is a result parameter which will contain the results of the search upon completion of the call. If no results are returned, *res is set to NULL.

Return Values

Table 9–23 SEARCH_ST Function Return Values

Value	Description
PLS_INTEGER	DBMS_LDAP.SUCCESS if the search operation succeeded. An exception is raised in all other cases.
res	If the search succeeded and there are entries, this parameter is set to a non-null value which can be used to iterate through the result set.

Exceptions

Table 9–24 SEARCH_ST Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_search_scope	Raised if the search scope is not one of SCOPE_BASE, SCOPE_ONELEVEL or SCOPE_SUBTREE.
invalid_search_time_value	Raised if the time value specified for the time out is invalid.
general_error	For all other errors. The error string associated with this exception will explain the error in detail.

Usage Notes

This function is very similar to DBMS_LDAP.search_s() except that it requires a time out value to be given.

See Also

DBMS_LDAP.search_s(), DBML_LDAP.first_entry(), DBMS_LDAP.next_entry.

FUNCTION first_entry

The function first_entry() is used to retrieve the first entry in the result set returned by either search_s() or search_st().

Syntax

```
FUNCTION first_entry
(
ld  IN SESSION,
msg IN MESSAGE
)
RETURN MESSAGE;
```

Parameters

Table 9–25 FIRST_ENTRY Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
msg	The search result, as obtained by a call to one of the synchronous search routines.

Return Values

Table 9–26 FIRST_ENTRY Return Values

Value	Description
MESSAGE	A handle to the first entry in the list of entries returned from the LDAP server. It is set to NULL if there was an error and an exception is raised.

Exceptions

Table 9–27 FIRST_ENTRY Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_message	Raised if the incoming msg handle is invalid.

Usage Notes

The function `first_entry()` should always be the first function used to retrieve the results from a search operation.

See Also

`DBMS_LDAP.next_entry()`, `DBMS_LDAP.search_s()`, `DBMS_LDAP.search_st()`

FUNCTION next_entry

The function `next_entry()` is used to iterate to the next entry in the result set of a search operation.

Syntax

```
FUNCTION next_entry
(
ld  IN SESSION,
msg IN MESSAGE
)
RETURN MESSAGE;
```

Parameters

Table 9–28 NEXT_ENTRY Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
msg	The search result, as obtained by a call to one of the synchronous search routines.

Return Values

Table 9–29 NEXT_ENTRY Function Return Values

Value	Description
MESSAGE	A handle to the next entry in the list of entries returned from the LDAP server. It is set to null if there was an error and an exception is raised.

Exceptions

Table 9–30 NEXT_ENTRY Function Exceptions

Exception	Description
invalid_session	Raised if the session handle, ld is invalid.
invalid_message	Raised if the incoming msg handle is invalid.

Usage Notes

The function next_entry() should always be called after a call to the function first_entry(). Also, the return value of a successful call to next_entry() should be used as msg argument used in a subsequent call to the function next_entry() to fetch the next entry in the list.

See Also

DBMS_LDAP.first_entry(), DBMS_LDAP.search_s(), DBMS_LDAP.search_st()

FUNCTION count_entries

This function is used to count the number of entries in the result set. It can also be used to count the number of entries remaining during a traversal of the result set using a combination of the functions first_entry() and next_entry().

Syntax

```
FUNCTION count_entries
(
ld  IN SESSION,
msg IN MESSAGE
)
RETURN PLS_INTEGER;
```

Parameters

Table 9–31 COUNT_ENTRY Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
msg	The search result, as obtained by a call to one of the synchronous search routines.

Return Values

Table 9–32 COUNT_ENTRY Function Return Values

Value	Description
PLS_INTEGER	Nonzero if there are entries in the result set. -1 if there was a problem.

Exceptions

Table 9–33 COUNT_ENTRY Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_message	Raised if the incoming msg handle is invalid.
count_entry_error	Raised if there was a problem in counting the entries.

Usage Notes

count_entries() returns the number of entries contained in a chain of entries; if an error occurs such as the res parameter being invalid, -1 is returned. The count_entries() call can also be used to count the number of entries that remain in a chain if called with a message, entry, or reference returned by first_message(), next_message(), first_entry(), next_entry(), first_reference(), next_reference().

See Also

DBMS_LDAP.first_entry(), DBMS_LDAP.next_entry().

FUNCTION first_attribute

The function first_attribute() fetches the first attribute of a given entry in the result set.

Syntax

```
FUNCTION first_attribute
(
ld          IN  SESSION,
ldapentry   IN  MESSAGE,
ber_elem    OUT BER_ELEMENT
)
RETURN VARCHAR2;
```

Parameters

Table 9–34 FIRST_ATTRIBUTE Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
ldapentry	The entry whose attributes are to be stepped through, as returned by first_entry() or next_entry().
ber_elem	A handle to a BER_ELEMENT that is used to keep track of attributes in the entry that have already been read.

Return Values

Table 9–35 FIRST_ATTRIBUTE Function Return Values

Value	Description
VARCHAR2	The name of the attribute if it exists. NULL if no attribute exists or if an error occurred.
ber_elem	A handle used by DBMS_LDAP.next_attribute() to iterate over all of the attributes

Exceptions

Table 9–36 FIRST_ATTRIBUTE Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_message	Raised if the incoming msg handle is invalid.

Usage Notes

The handle to the BER_ELEMENT returned as a function parameter to first_attribute() should be used in the next call to next_attribute() to iterate through the various attributes of an entry. The name of the attribute returned from a call to first_attribute() can in turn be used in calls to the functions get_values() or get_values_len() to get the values of that particular attribute.

See Also

DBMS_LDAP.next_attribute(), DBMS_LDAP.get_values(), DBMS_LDAP.get_values_len(), DBMS_LDAP.first_entry(), DBMS_LDAP.next_entry().

FUNCTION next_attribute

The function next_attribute() retrieves the next attribute of a given entry in the result set.

Syntax

```
FUNCTION next_attribute
(
ld       IN SESSION,
ldapentry      IN MESSAGE,
ber_elem IN BER_ELEMENT
)
RETURN VARCHAR2;
```

Parameters

Table 9–37 NEXT_ATTRIBUTE Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
ldapentry	The entry whose attributes are to be stepped through, as returned by first_entry() or next_entry().
ber_elem	A handle to a BER_ELEMENT that is used to keep track of attributes in the entry that have been read.

Return Values

Table 9–38 NEXT_ATTRIBUTE Function Return Values

Value	Description
VARCHAR2 (function return)	The name of the attribute if it exists.

Exceptions

Table 9–39 NEXT_ATTRIBUTE Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_message	Raised if the incoming msg handle is invalid.

Usage Notes

The handle to the BER_ELEMENT returned as a function parameter to first_attribute() should be used in the next call to next_attribute() to iterate through the various attributes of an entry. The name of the attribute returned from a call to next_attribute() can in turn be used in calls to the functions get_values() or get_values_len() to get the values of that particular attribute.

See Also

DBMS_LDAP.first_attribute(), DBMS_LDAP.get_values(), DBMS_LDAP.get_values_len(), DBMS_LDAP.first_entry(), DBMS_LDAP.next_entry().

FUNCTION get_dn

The function get_dn() retrieves the X.500 distinguished name of given entry in the result set.

Syntax

```
FUNCTION get_dn
(
ld  IN SESSION,
ldapentrymsg IN MESSAGE
)
RETURN VARCHAR2;
```

Parameters

Table 9–40 GET_DN Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
ldapentry	The entry whose DN is to be returned.

Return Values

Table 9–41 GET_DN Function Return Values

Value	Description
VARCHAR2	The X.500 Distinguished name of the entry as a PL/SQL string. NULL if there was a problem.

Exceptions

Table 9–42 GET_DN Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_message	Raised if the incoming msg handle is invalid.
get_dn_error	Raised if there was a problem in determining the DN.

Usage Notes

The function `get_dn()` can be used to retrieve the DN of an entry as the program logic is iterating through the result set. This can in turn be used as an input to `explode_dn()` to retrieve the individual components of the DN.

See Also

`DBMS_LDAP.explode_dn()`.

FUNCTION get_values

The function `get_values()` can be used to retrieve all of the values associated with a given attribute in a given entry.

Syntax

```
FUNCTION get_values
(
ld   IN SESSION,
ldapentry IN MESSAGE,
attr IN VARCHAR2
)
RETURN STRING_COLLECTION;
```

Parameters

Table 9–43 GET_VALUES Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
ldapentry	A valid handle to an entry returned from a search result.
attr	The name of the attribute for which values are being sought.

Return Values

Table 9–44 GET_VALUES Function Return Values

Value	Description
STRING_COLLECTION	A PL/SQL string collection containing all of the values of the given attribute. NULL if there are no values associated with the given attribute.

Exceptions

Table 9–45 GET_VALUES Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_message	Raised if the incoming entry handle is invalid.

Usage Notes

The function get_values() can only be called after the handle to entry has been first retrieved by call to either first_entry() or next_entry(). The name of the attribute may be known beforehand or can be determined by a call to first_attribute() or next_attribute().The function get_values() always assumes that the data type of the attribute it is retrieving is a string. For retrieving binary data types, get_values_len() should be used.

See Also

DBMS_LDAP.first_entry(), DBMS_LDAP.next_entry(), DBMS_LDAP.count_values(), DBMS_LDAP.get_values_len().

FUNCTION get_values_len

The function get_values_len() can be used to retrieve values of attributes that have a binary syntax.

Syntax

```
FUNCTION get_values_len
(
ld   IN SESSION,
ldapentry IN MESSAGE,
attr IN VARCHAR2
)
RETURN BINVAL_COLLECTION;
```

Parameters

Table 9–46 GET_VALUES_LEN Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
ldapentrymsg	A valid handle to an entry returned from a search result.
attr	The string name of the attribute for which values are being sought.

Return Values

Table 9–47 GET_VALUES_LEN Function Return Values

Value	Description
BINVAL_COLLECTION	A PL/SQL 'Raw' collection containing all the values of the given attribute. NULL if there are no values associated with the given attribute.

Exceptions

Table 9–48 GET_VALUES_LEN Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_message	Raised if the incoming entry handle is invalid.

Usage Notes

The function get_values_len() can only be called after the handle to an entry has been retrieved by a call to either first_entry() or next_entry().The name of the attribute may be known beforehand or can also be determined by a call to first_attribute() or next_attribute().This function can be used to retrieve both binary and non-binary attribute values.

See Also

DBMS_LDAP.first_entry(), DBMS_LDAP.next_entry(), DBMS_LDAP.count_values_len(), DBMS_LDAP.get_values().

FUNCTION delete_s

The function delete_s() can be used to remove a leaf entry in the DIT.

Syntax

```
FUNCTION delete_s
(
ld      IN SESSION,
entrydn IN VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters

Table 9–49 DELETE_S Function Parameters

Parameter Name	Description
ld	A valid LDAP session.
entrydn	The X.500 distinguished name of the entry to delete.

Return Values

Table 9–50 DELETE_S Function Return Values

Value	Description
PLS_INTEGER	DBMS_LDAP.SUCCESS if the delete operation was successful. An exception is raised otherwise.

Exceptions

Table 9–51 DELETE_S Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_entry_dn	Raised if the distinguished name of the entry is invalid.
general_error	For all other errors. The error string associated with this exception will explain the error in detail.

Usage Notes

The function delete_s() can be used to remove only leaf entries in the DIT. A leaf entry is an entry that does not have any entries under it. This function cannot be used to delete non-leaf entries.

See Also

DBMS_LDAP.modrdn2_s().

FUNCTION modrdn2_s

The function modrdn2_s() can be used to rename the relative distinguished name of an entry.

Syntax

```
FUNCTION modrdn2_s
(
ld IN SESSION,
entrydn in VARCHAR2
newrdn in VARCHAR2
deleteoldrdn IN PLS_INTEGER
)
RETURN PLS_INTEGER;
```

Parameters

Table 9–52 MODRDN2_S Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
entrydn	The distinguished name of the entry (This entry must be a leaf node in the DIT.).
newrdn	The new relative distinguished name of the entry.
deleteoldrdn	A boolean value that, if nonzero, indicates that the attribute values from the old name should be removed from the entry.

Return Values

Table 9–53 MODRDN2_S Function Return Values

Value	Description
PLS_INTEGER	DBMS_LDAP.SUCCESS if the operation was successful. An exception is raised otherwise.

Exceptions

Table 9–54 MODRDN2_S Function Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_entry_dn	Raised if the distinguished name of the entry is invalid.
invalid_rdn	Invalid LDAP RDN.
invalid_deleteoldrdn	Invalid LDAP deleteoldrdn.
general_error	For all other errors. The error string associated with this exception will explain the error in detail.

Usage Notes

The function nodrdn2_s() can be used to rename the leaf nodes of a DIT. It simply changes the relative distinguished name by which they are known. The use of this function is being deprecated in the LDAP v3 standard. Please use rename_s(), which fulfills the same purpose.

See Also

DBMS_LDAP.rename_s().

FUNCTION err2string

The function err2string() can be used to convert an LDAP error code to a string in the local language in which the API is operating.

Syntax

```
FUNCTION err2string
(
ldap_err IN PLS_INTEGER
)
RETURN VARCHAR2;
```

Parameters

Table 9–55 ERR2STRING Function Parameters

Parameter	Description
ldap_err	An error number returned from one of the API calls.

Return Values

Table 9–56 ERR2STRING Function Return Values

Value	Description
VARCHAR2	A character string translated to the local language. The string describes the error in detail.

Exceptions

`err2string()` raises no exceptions.

Usage Notes

In this release, the exception handling mechanism automatically invokes this function if any of the API calls encounter an error.

FUNCTION create_mod_array

The function `create_mod_array()` allocates memory for array modification entries that are applied to an entry using the `modify_s()` or `add_s()` functions.

Syntax

```
FUNCTION create_mod_array
(
num IN PLS_INTEGER
)
RETURN MOD_ARRAY;
```

Parameters

Table 9–57 CREATE_MOD_ARRAY Function Parameters

Parameter	Description
num	The number of the attributes that you want to add or modify.

Return Values

Table 9–58 CREATE_MOD_ARRAY Function Return Values

Value	Description
MOD_ARRAY	The data structure holds a pointer to an LDAP mod array. Returns `NULL` if there was a problem.

Exceptions

`create_mod_array()` raises no exceptions.

Usage Notes

This function is one of the preparation steps for `DBMS_LDAP.add_s` and `DBMS_LDAP.modify_s`. It calls `DBMS_LDAP.free_mod_array` to free memory after the calls to `add_s` or `modify_s` have completed.

See Also

`DBMS_LDAP.populate_mod_array()`, `DBMS_LDAP.modify_s()`, `DBMS_LDAP.add_s()`, and `DBMS_LDAP.free_mod_array()`.

PROCEDURE populate_mod_array (String Version)

Populates one set of attribute information for add or modify operations.

Syntax

```
PROCEDURE populate_mod_array
(
modptr   IN DBMS_LDAP.MOD_ARRAY,
mod_op   IN PLS_INTEGER,
mod_type IN VARCHAR2,
modval   IN DBMS_LDAP.STRING_COLLECTION
);
```

Parameters

Table 9–59 POPULATE_MOD_ARRAY (String Version) Procedure Parameters

Parameter	Description
modptr	The data structure holds a pointer to an LDAP mod array.
mod_op	This field specifies the type of modification to perform.
mod_type	This field indicates the name of the attribute type to which the modification applies.
modval	This field specifies the attribute values to add, delete, or replace. It is for string values only.

Exceptions

Table 9–60 POPULATE_MOD_ARRAY (String Version) Procedure Exceptions

Exception	Description
invalid_mod_array	Invalid LDAP mod array
invalid_mod_option	Invalid LDAP mod option
invalid_mod_type	Invalid LDAP mod type
invalid_mod_value	Invalid LDAP mod value

Usage Notes

This function is one of the preparation steps for DBMS_LDAP.add_s and DBMS_LDAP.modify_s. It has to happen after DBMS_LDAP.create_mod_array is called.

See Also

DBMS_LDAP.create_mod_array(), DBMS_LDAP.modify_s(), DBMS_LDAP.add_s(), and DBMS_LDAP.free_mod_array().

PROCEDURE populate_mod_array (Binary Version)

Populates one set of attribute information for add or modify operations. This procedure call occurs after DBMS_LDAP.create_mod_array() is called.

Syntax

```
PROCEDURE populate_mod_array
(
modptr   IN DBMS_LDAP.MOD_ARRAY,
mod_op   IN PLS_INTEGER,
```

```
mod_type IN VARCHAR2,
modbval  IN DBMS_LDAP.BERVAL_COLLECTION
);
```

Parameters

Table 9–61 POPULATE_MOD_ARRAY (Binary Version) Procedure Parameters

Parameter	Description
modptr	This data structure holds a pointer to an LDAP mod array.
mod_op	This field specifies the type of modification to perform.
mod_type	This field indicates the name of the attribute type to which the modification applies.
modbval	This field specifies the attribute values to add, delete, or replace. It is for the binary values.

Exceptions

Table 9–62 POPULATE_MOD_ARRAY (Binary Version) Procedure Exceptions

Exception	Description
invalid_mod_array	Invalid LDAP mod array.
invalid_mod_option	Invalid LDAP mod option.
invalid_mod_type	Invalid LDAP mod type.
invalid_mod_value	Invalid LDAP mod value.

Usage Notes

This function is one of the preparation steps for DBMS_LDAP.add_s and DBMS_LDAP.modify_s. It is invoked after DBMS_LDAP.create_mod_array is called.

See Also

DBMS_LDAP.create_mod_array(), DBMS_LDAP.modify_s(), DBMS_LDAP.add_s(), and DBMS_LDAP.free_mod_array().

PROCEDURE populate_mod_array (Binary Version. Uses BLOB Data Type)

Populates one set of attribute information for add or modify operations. This procedure call occurs after DBMS_LDAP.create_mod_array() is called.

Syntax

```
PROCEDURE populate_mod_array
(
modptr IN DBMS_LDAP.MOD_ARRAY,
mod_op IN PLS_INTEGER,
mod_type IN VARCHAR2,
modbval IN DBMS_LDAP.BLOB_COLLECTION
);
```

Parameters

Table 9–63 POPULATE_MOD_ARRAY (Binary) Parameters

Parameter	Description
modptr	This data structure holds a pointer to an LDAP mod array.
mod_op	This field specifies the type of modification to perform.
mod_type	This field indicates the name of the attribute type to which the modification applies.
modbval	This field specifies the binary attribute values to add, delete, or replace.

Exceptions

Table 9–64 POPULATE_MOD_ARRAY (Binary) Exceptions

Exception	Description
invalid_mod_array	Invalid LDAP mod array.
invalid_mod_option	Invalid LDAP mod option.
invalid_mod_type	Invalid LDAP mod type.
invalid_mod_value	Invalid LDAP mod value.

Usage Notes

This function is one of the preparation steps for DBMS_LDAP.add_s and DBMS_LDAP.modify_s. It is invoked after DBMS_LDAP.create_mod_array is called.

See Also

DBMS_LDAP.create_mod_array(), DBMS_LDAP.modify_s(), DBMS_LDAP.add_s(), and DBMS_LDAP.free_mod_array().

FUNCTION get_values_blob

The function get_values_blob() can be used to retrieve larger values of attributes that have a binary syntax.

Syntax

```
Syntax
FUNCTION get_values_blob
(
ld IN SESSION,
ldapentry IN MESSAGE,
attr IN VARCHAR2
)
RETURN BLOB_COLLECTION;
```

Parameters

Table 9–65 GET_VALUES_BLOB Parameters

Parameter	Description
ld	A valid LDAP session handle.
ldapentrymsg	A valid handle to an entry returned from a search result.

Table 9–65 (Cont.) GET_VALUES_BLOB Parameters

Parameter	Description
attr	The string name of the attribute for which values are being sought.

Return Values

Table 9–66 get_values_blob Return Values

Value	Description
BLOB_COLLECTION	A PL/SQL BLOB collection containing all the values of the given attribute.
NULL	No values are associated with the given attribute.

Exceptions

Table 9–67 get_values_blob Exceptions

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid message	Raised if the incoming entry handle is invalid.

Usage Notes

The function get_values_blob() can only be called after the handle to an entry has been retrieved by a call to either first_entry() or next_entry(). The name of the attribute may be known beforehand or can also be determined by a call to first_attribute() or next_attribute(). This function can be used to retrieve both binary and nonbinary attribute values.

See Also

DBMS_LDAP.first_entry(), DBMS_LDAP.next_entry(), DBMS_LDAP.count_values_blob(), DBMS_LDAP.get_values().

FUNCTION count_values_blob

Counts the number of values returned by DBMS_LDAP.get_values_blob().

Syntax

```
FUNCTION count_values_blob
(
values IN DBMS_LDAP.BLOB_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

Table 9–68 COUNT_VALUES_BLOB Parameters

Parameter	Description
values	The collection of large binary values.

Return Values

Table 9–69 COUNT_VALUES_BLOB Return Values

Values	Description
PLS_INTEGER	Indicates the success or failure of the operation.

Exceptions

The function `count_values_blob()` raises no exceptions.

See Also

`DBMS_LDAP.count_values()`, `DBMS_LDAP.get_values_blob()`.

FUNCTION value_free_blob

Frees the memory associated with `BLOB_COLLECTION` returned by `DBMS_LDAP.get_values_blob()`.

Syntax

```
PROCEDURE value_free_blob
(
vals IN OUT DBMS_LDAP.BLOB_COLLECTION
);
```

Parameters

Table 9–70 VALUE_FREE_BLOB Parameters

Parameter	Description
vals	The collection of large binary values returned by `DBMS_LDAP.get_values_blob()`.

Exceptions

`value_free_blob()` raises no exceptions.

See Also

`DBMS_LDAP.get_values_blob()`.

FUNCTION modify_s

Performs a synchronous modification of an existing LDAP directory entry.

Syntax

```
FUNCTION modify_s
(
ld      IN DBMS_LDAP.SESSION,
entrydn IN VARCHAR2,
modptr  IN DBMS_LDAP.MOD_ARRAY
)
RETURN PLS_INTEGER;
```

Parameters

Table 9–71 MODIFY_S Function Parameters

Parameter	Description
ld	This parameter is a handle to an LDAP session returned by a successful call to DBMS_LDAP.init().
entrydn	This parameter specifies the name of the directory entry whose contents are to be modified.
modptr	This parameter is the handle to an LDAP mod structure, as returned by successful call to DBMS_LDAP.create_mod_array().

Return Values

Table 9–72 MODIFY_S Function Return Values

Value	Description
PLS_INTEGER	Indicates the success or failure of the modification operation.

Exceptions

Table 9–73 MODIFY_S Function Exceptions

Exception	Description
invalid_session	Invalid LDAP session.
invalid_entry_dn	Invalid LDAP entry dn.
invalid_mod_array	Invalid LDAP mod array.

Usage Notes

This function call has to follow successful calls of DBMS_LDAP.create_mod_array() and DBMS_LDAP.populate_mod_array().

See Also

DBMS_LDAP.create_mod_array(),DBMS_LDAP.populate_mod_array(), DBMS_LDAP.add_s(), and DBMS_LDAP.free_mod_array().

FUNCTION add_s

Adds a new entry to the LDAP directory synchronously. Before calling add_s, DBMS_LDAP.create_mod_array() and DBMS_LDAP.populate_mod_array() must be called.

Syntax

```
FUNCTION add_s
(
ld      IN DBMS_LDAP.SESSION,
entrydn IN VARCHAR2,
modptr  IN DBMS_LDAP.MOD_ARRAY
)
RETURN PLS_INTEGER;
```

Parameters

Table 9–74 ADD_S Function Parameters

Parameter	Description
ld	This parameter is a handle to an LDAP session, as returned by a successful call to DBMS_LDAP.init().
entrydn	This parameter specifies the name of the directory entry to be created.
modptr	This parameter is the handle to an LDAP mod structure, as returned by successful call to DBMS_LDAP.create_mod_array().

Return Values

Table 9–75 ADD_S Function Return Values

Value	Description
PLS_INTEGER	Indicates the success or failure of the modification operation.

Exceptions

Table 9–76 ADD_S Function Exceptions

Exception	Description
invalid_session	Invalid LDAP session.
invalid_entry_dn	Invalid LDAP entry dn.
invalid_mod_array	Invalid LDAP mod array.

Usage Notes

The parent entry of the entry to be added must already exist in the directory. This function call has to follow successful calls to DBMS_LDAP.create_mod_array() and DBMS_LDAP.populate_mod_array().

See Also

DBMS_LDAP.create_mod_array(), DBMS_LDAP.populate_mod_array(), DBMS_LDAP.modify_s(), and DBMS_LDAP.free_mod_array().

PROCEDURE free_mod_array

Frees the memory allocated by DBMS_LDAP.create_mod_array().

Syntax

```
PROCEDURE free_mod_array
(
modptr IN DBMS_LDAP.MOD_ARRAY
);
```

Parameters

Table 9–77 FREE_MOD_ARRAY Procedure Parameters

Parameter	Description
`modptr`	This parameter is the handle to an LDAP mod structure returned by a successful call to `DBMS_LDAP.create_mod_array()`.

Exceptions

`free_mod_array` raises no exceptions.

See Also

`DBMS_LDAP.populate_mod_array()`, `DBMS_LDAP.modify_s()`, `DBMS_LDAP.add_s()`, and `DBMS_LDAP.create_mod_array()`.

FUNCTION count_values

Counts the number of values returned by `DBMS_LDAP.get_values()`.

Syntax

```
FUNCTION count_values
(
values IN DBMS_LDAP.STRING_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

Table 9–78 COUNT_VALUES Function Parameters

Parameter	Description
`values`	The collection of string values.

Return Values

Table 9–79 COUNT_VALUES Function Return Values

Value	Description
`PLS_INTEGER`	Indicates the success or failure of the operation.

Exceptions

`count_values` raises no exceptions.

See Also

`DBMS_LDAP.count_values_len()`, `DBMS_LDAP.get_values()`.

FUNCTION count_values_len

Counts the number of values returned by `DBMS_LDAP.get_values_len()`.

Syntax

```
FUNCTION count_values_len
(
```

```
values IN DBMS_LDAP.BINVAL_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

Table 9–80 COUNT_VALUES_LEN Function Parameters

Parameter	Description
values	The collection of binary values.

Return Values

Table 9–81 COUNT_VALUES_LEN Function Return Values

Value	Description
PLS_INTEGER	Indicates the success or failure of the operation.

Exceptions

count_values_len raises no exceptions.

See Also

DBMS_LDAP.count_values(), DBMS_LDAP.get_values_len().

FUNCTION rename_s

Renames an LDAP entry synchronously.

Syntax

```
FUNCTION rename_s
(
ld           IN SESSION,
dn           IN VARCHAR2,
newrdn       IN VARCHAR2,
newparent    IN VARCHAR2,
deleteoldrdn IN PLS_INTEGER,
serverctrls  IN LDAPCONTROL,
clientctrls  IN LDAPCONTROL
)
RETURN PLS_INTEGER;
```

Parameters

Table 9–82 RENAME_S Function Parameters

Parameter	Description
ld	This parameter is a handle to an LDAP session returned by a successful call to DBMS_LDAP.init().
dn	This parameter specifies the name of the directory entry to be renamed or moved.
newrdn	This parameter specifies the new RDN.
newparent	This parameter specifies the DN of the new parent.
deleteoldrdn	This parameter specifies whether the old RDN should be retained. If this value is 1, the old RDN is removed.

Table 9–82 (Cont.) RENAME_S Function Parameters

Parameter	Description
serverctrls	Currently not supported.
clientctrls	Currently not supported.

Return Values

Table 9–83 RENAME_S Function Return Values

Value	Description
PLS_INTEGER	The indication of the success or failure of the operation.

Exceptions

Table 9–84 RENAME_S Function Exceptions

Exception	Description
invalid_session	Invalid LDAP Session.
invalid_entry_dn	Invalid LDAP DN.
invalid_rdn	Invalid LDAP RDN.
invalid_newparent	Invalid LDAP newparent.
invalid_deleteoldrdn	Invalid LDAP deleteoldrdn.

See Also

DBMS_LDAP.modrdn2_s().

FUNCTION explode_dn

Breaks a DN up into its components.

Syntax

```
FUNCTION explode_dn
(
dn      IN VARCHAR2,
notypes IN PLS_INTEGER
)
RETURN STRING_COLLECTION;
```

Parameters

Table 9–85 EXPLODE_DN Function Parameters

Parameter	Description
dn	This parameter specifies the name of the directory entry to be broken up.
notypes	This parameter specifies whether the attribute tags will be returned. If this value is not 0, no attribute tags are returned.

Return Values

Table 9–86 EXPLODE_DN Function Return Values

Value	Description
STRING_COLLECTION	An array of strings. If the DN cannot be broken up, NULL will be returned.

Exceptions

Table 9–87 EXPLODE_DN Function Exceptions

Exception	Description
invalid_entry_dn	Invalid LDAP DN.
invalid_notypes	Invalid LDAP notypes value.

See Also

DBMS_LDAP.get_dn().

FUNCTION open_ssl

Establishes an SSL (Secure Sockets Layer) connection over an existing LDAP connection.

Syntax

```
FUNCTION open_ssl
(
ld              IN SESSION,
sslwrl          IN VARCHAR2,
sslwalletpasswd IN VARCHAR2,
sslauth         IN PLS_INTEGER
)
RETURN PLS_INTEGER;
```

Parameters

Table 9–88 OPEN_SSL Function Parameters

Parameter	Description
ld	This parameter is a handle to an LDAP session that is returned by a successful call to DBMS_LDAP.init().
sslwrl	This parameter specifies the wallet location. Required for one-way or two-way SSL connections.
sslwalletpasswd	This parameter specifies the wallet password. Required for one-way or two-way SSL connections.
sslauth	This parameter specifies the SSL Authentication Mode. (1 for no authentication, 2 for one-way authentication required, 3 for two-way authentication).

Return Values

Table 9–89 OPEN_SSL Function Return Values

Value	Description
PLS_INTEGER	Indicates the success or failure of the operation.

Exceptions

Table 9–90 OPEN_SSL Function Exceptions

Exception	Description
invalid_session	Invalid LDAP Session.
invalid_ssl_wallet_loc	Invalid LDAP SSL wallet location.
invalid_ssl_wallet_passwd	Invalid LDAP SSL wallet password.
invalid_ssl_auth_mode	Invalid LDAP SSL authentication mode.

Usage Notes

Need to call DBMS_LDAP.init() first to acquire a valid ldap session.

See Also

DBMS_LDAP.init().

FUNCTION msgfree

This function frees the chain of messages associated with the message handle returned by synchronous search functions.

Syntax

```
FUNCTION msgfree
(
res             IN MESSAGE
)
RETURN PLS_INTEGER;
```

Parameters

Table 9–91 MSGFREE Function Parameters

Parameter	Description
res	The message handle obtained by a call to one of the synchronous search routines.

Return Values

Table 9–92 MSGFREE Return Values

Value	Description
PLS_INTEGER	Indicates the type of the last message in the chain. The function might return any of the following values: ■ DBMS_LDAP.LDAP_RES_BIND ■ DBMS_LDAP.LDAP_RES_SEARCH_ENTRY ■ DBMS_LDAP.LDAP_RES_SEARCH_REFERENCE ■ DBMS_LDAP.LDAP_RES_SEARCH_RESULT ■ DBMS_LDAP.LDAP_RES_MODIFY ■ DBMS_LDAP.LDAP_RES_ADD ■ DBMS_LDAP.LDAP_RES_DELETE ■ DBMS_LDAP.LDAP_RES_MODDN ■ DBMS_LDAP.LDAP_RES_COMPARE ■ DBMS_LDAP.LDAP_RES_EXTENDED

Exceptions

msgfree raises no exceptions.

See Also

DBMS_LDAP.search_s(), DBMS_LDAP.search_st().

FUNCTION ber_free

This function frees the memory associated with a handle to BER ELEMENT.

Syntax

```
FUNCTION ber_free
(
ber_elem IN BER_ELEMENT,
freebuf  IN PLS_INTEGER
)
```

Parameters

Table 9–93 BER_FREE Function Parameters

Parameter	Description
ber_elem	A handle to BER ELEMENT.
freebuf	The value of this flag should be 0 while the BER ELEMENT returned from DBMS_LDAP.first_attribute() is being freed. For any other case, the value of this flag should be 1. The default value of this parameter is zero.

Return Values

ber_free returns no values.

Exceptions

ber_free raises no exceptions.

See Also

DBMS_LDAP.first_attribute(),DBMS_LDAP.next_attribute().

FUNCTION nls_convert_to_utf8

The nls_convert_to_utf8() function converts the input string containing database character set data to UTF8 character set data and returns it.

Syntax

```
Function nls_convert_to_utf8
(
data_local IN VARCHAR2
)
RETURN VARCHAR2;
```

Parameters

Table 9–94 Parameters for nls_convert_to_utf8

Parameter	Description
data_local	Contains the database character set data.

Return Values

Table 9–95 Return Values for nls_convert_to_utf8

Value	Description
VARCHAR2	UTF8 character set data string.

Usage Notes

The functions in DBMS_LDAP package expect the input data to be UTF8 character set data if the UTF8_CONVERSION package variable is set to FALSE. The nls_convert_to_utf8() function converts database character set data to UTF8 character set data.

If the UTF8_CONVERSION package variable of the DBMS_LDAP package is set to TRUE, functions in the DBMS_LDAP package expect input data to be database character set data.

See Also

DBMS_LDAP.nls_convert_from_utf8(), DBMS_LDAP.nls_get_dbcharset_name().

FUNCTION nls_convert_to_utf8

The nls_convert_to_utf8() function converts the input string collection containing database character set data to UTF8 character set data. It then returns the converted data.

Syntax

```
Function nls_convert_to_utf8
(
data_local IN STRING_COLLECTION
)
RETURN STRING_COLLECTION;
```

Parameters

Table 9–96 Parameters for nls_convert_to_utf8

Parameter	Description
data_local	Collection of strings containing database character set data.

Return Values

Table 9–97 Return Values for nls_convert_to_utf8

Value	Description
STRING_COLLECTION	Collection of strings containing UTF8 character set data.

Usage Notes

The functions in the DBMS_LDAP package expect the input data to be in the UTF8 character set if the UTF8_CONVERSION package variable is set to FALSE. The nls_convert_to_utf8() function converts the input data from the database character set to the UTF8 character set.

If the UTF8_CONVERSION package variable of the DBMS_LDAP package is set to TRUE, functions in the DBMS_LDAP package expect the input data to be in the database character set.

See Also

DBMS_LDAP.nls_convert_from_utf8(), DBMS_LDAP.nls_get_dbcharset_name().

FUNCTION nls_convert_from_utf8

The nls_convert_from_utf8() function converts the input string containing UTF8 character set to database character set data. It then returns this data.

Syntax

```
Function nls_convert_from_utf8
(
data_utf8 IN VARCHAR2
)
RETURN VARCHAR2;
```

Parameters

Table 9–98 Parameter for nls_convert_from_utf8

Parameter	Description
data_utf8	Contains UTF8 character set data.

Return Values

Table 9–99 Return Value for nls_convert_from_utf8

Value	Description
VARCHAR2	Data string in the database character set.

Usage Notes

The functions in the DBMS_LDAP package return UTF8 character set data if the UTF8_CONVERSION package variable is set to FALSE. The nls_convert_from_utf8() function converts the output data from the UTF8 character set to the database character set.

If the UTF8_CONVERSION package variable of the DBMS_LDAP package is set to TRUE, functions in the DBMS_LDAP package return database character set data.

See Also

DBMS_LDAP.nls_convert_to_utf8(), DBMS_LDAP.nls_get_dbcharset_name().

FUNCTION nls_convert_from_utf8

The nls_convert_from_utf8() function converts the input string collection containing UTF8 character set data to database character set data. It then returns this data.

Syntax

```
Function nls_convert_from_utf8
(
data_utf8 IN STRING_COLLECTION
)
RETURN STRING_COLLECTION;
```

Parameters

Table 9–100 Parameter for nls_convert_from_utf8

Parameter	Description
data_utf8	Collection of strings containing UTF8 character set data.

Return Values

Table 9–101 Return Value for nls_convert_from_utf8

Value	Description
VARCHAR2	Collection of strings containing database character set data.

Usage Notes

The functions in the DBMS_LDAP package return UTF8 character set data if the UTF8_CONVERSION package variable is set to FALSE. nls_convert_from_utf8() converts the output data from the UTF8 character set to the database character set. If the UTF8_CONVERSION package variable of the DBMS_LDAP package is set to TRUE, functions in the DBMS_LDAP package return database character set data.

See Also

DBMS_LDAP.nls_convert_to_utf8(), DBMS_LDAP.nls_get_dbcharset_name().

FUNCTION nls_get_dbcharset_name

The `nls_get_dbcharset_name()` function returns a string containing the database character set name.

Syntax

```
Function nls_get_dbcharset_name

RETURN VARCHAR2;
```

Parameters

None.

Return Values

Table 9–102 Return Value for nls_get_dbcharset_name

Value	Description
VARCHAR2	String containing the database character set name.

See Also

`DBMS_LDAP.nls_convert_to_utf8()`, `DBMS_LDAP.nls_convert_from_utf8()`.

10

Java API Reference

The standard Java APIs for Oracle Internet Directory are available as the Java Naming and Directory Interface (JNDI) from Sun Microsystems. The JNDI is found at this link:

`http://java.sun.com/products/jndi`

The Oracle extensions to the standard APIs are found in *Oracle Internet Directory API Reference*.

Sample code for the Java APIs is available at this URL:

`http://www.oracle.com/technology/sample_code/id_mgmt`

11

DBMS_LDAP_UTL PL/SQL Reference

This chapter contains reference material for the DBMS_LDAP_UTL package, which contains Oracle Extension utility functions. The chapter contains these topics:

- Summary of Subprograms
- Subprograms
- Function Return Code Summary
- Data Type Summary

> **Note:** Sample code for the DBMS_LDAP_UTL package is available at this URL:
>
> http://www.oracle.com/technology/sample_code/id_mgmt

Summary of Subprograms

Table 11–1 DBMS_LDAP_UTL User-Related Subprograms

Function or Procedure	Purpose
Function authenticate_user	Authenticates a user against an LDAP server.
Function create_user_handle	Creates a user handle.
Function set_user_handle_properties	Associates the given properties to the user handle.
Function get_user_properties	Retrieves user properties from an LDAP server.
Function set_user_properties	Modifies the properties of a user.
Function get_user_extended_properties	Retrieves user extended properties.
Function get_user_dn	Retrieves a user DN.
Function check_group_membership	Checks whether a user is member of a given group.
Function locate_subscriber_for_user	Retrieves the subscriber for the given user.
Function get_group_membership	Retrieves a list of groups of which the user is a member.

Table 11–2 DBMS_LDAP_UTL Group-Related Subprograms

Function or Procedure	Purpose
Function create_group_handle	Creates a group handle.
Function set_group_handle_properties	Associates the given properties with the group handle.
Function get_group_properties	Retrieves group properties from an LDAP server.
Function get_group_dn	Retrieves a group DN.

Table 11–3 DBMS_LDAP_UTL Subscriber-Related Subprograms

Function or Procedure	Purpose
Function create_subscriber_handle	Creates a subscriber handle.
Function get_subscriber_properties	Retrieves subscriber properties from an LDAP server.
Function get_subscriber_dn	Retrieves a subscriber DN.

Table 11–4 DBMS_LDAP_UTL Miscellaneous Subprograms

Function or Procedure	Purpose
Function normalize_dn_with_case	Normalizes the DN string.
Function get_property_names	Retrieves a list of property names in a `PROPERTY_SET`.
Function get_property_values	Retrieves a list of values for a property name.
Function get_property_values_blob	Retrieves a list of large binary values for a property name.
Procedure property_value_free_blob	Frees the memory associated with `BLOB_COLLECTION` returned by `DBMS_LDAP_UTL.get_property_values_blob()`.
Function get_property_values_len	Retrieves a list of binary values for a property name.
Procedure free_propertyset_collection	Frees `PROPERTY_SET_COLLECTION`.
Function create_mod_propertyset	Creates a `MOD_PROPERTY_SET`.
Function populate_mod_propertyset	Populates a `MOD_PROPERTY_SET` structure.
Procedure free_mod_propertyset	Frees a `MOD_PROPERTY_SET`.
Procedure free_handle	Frees handles.
Function check_interface_version	Checks for support of the interface version.

Subprograms

This section contains the following topics:

User-Related Subprograms

A user is represented by the DBMS_LDAP_UTL.HANDLE data type. You can create a user handle by using a DN, GUID, or simple name, along with the appropriate subscriber handle. When a simple name is used, additional information from the root Oracle Context and the subscriber Oracle Context is used to identify the user. This example shows a user handle being created:

```
retval := DBMS_LDAP_UTL.create_user_handle(
user_handle,
DBMS_LDAP_UTL.TYPE_DN,
"cn=user1,cn=users,o=acme,dc=com"
);
```

This user handle must be associated with an appropriate subscriber handle. If, for example, `subscriber_handle` is `o=acme,dc=com`, the subscriber handle can be associated in the following way:

```
retval := DBMS_LDAP_UTL.set_user_handle_properties(
user_handle,
DBMS_LDAP_UTL.SUBSCRIBER_HANDLE,
subscriber_handle
);
```

Common uses of user handles include setting and getting user properties and authenticating the user. Here is a handle that authenticates a user:

```
retval := DBMS_LDAP_UTL.authenticate_user(
my_session
user_handle
DBMS_LDAP_UTL.AUTH_SIMPLE,
"welcome"
NULL
);
```

In this example, the user is authenticated using a clear text password `welcome`.

Here is a handle that retrieves a user's telephone number:

```
--my_attrs is of type DBMS_LDAP.STRING_COLLECTION
my_attrs(1) := 'telephonenumber';
retval := DBMS_LDAP_UTL.get_user_properties(
my_session,
my_attrs,
DBMS_LDAP_UTL.ENTRY_PROPERTIES,
my_pset_coll
);
```

See Also: "DBMS_LDAP_UTL Sample Code" on page B-9 for more examples of user handles

Function authenticate_user

The function `authenticate_user()`authenticates the user against Oracle Internet Directory.

Syntax

```
FUNCTION authenticate_user
(
ld IN SESSION,
user_handle IN HANDLE,
auth_type IN PLS_INTEGER,
credentials IN VARCHAR2,
binary_credentials IN RAW
)
RETURN PLS_INTEGER;
```

Parameters

Table 11–5 AUTHENTICATE_USER Function Parameters

Parameter Name	Parameter Type	Parameter Description
`ld`	`SESSION`	A valid LDAP session handle.
`user_handle`	`HANDLE`	The user handle.
`auth_type`	`PLS_INTEGER`	Type of authentication. The only valid value is `DBMS_LDAP_UTL.AUTH_SIMPLE`
`credentials`	`VARCHAR2`	The user credentials.
`binary_credentials`	`RAW`	The binary credentials. This parameter is optional. It can be `NULL` by default.

Return Values

Table 11–6 AUTHENTICATE_USER Function Return Values

Value	Description
`DBMS_LDAP_UTL.SUCCESS`	On a successful completion.
`DBMS_LDAP_UTL.PARAM_ERROR`	Invalid input parameters.
`DBMS_LDAP_UTL.GENERAL_ERROR`	Authentication failed.
`DBMS_LDAP_UTL.NO_SUCH_USER`	User does not exist.
`DBMS_LDAP_UTL.MULTIPLE_USER_ENTRIES`	The user has multiple DN entries.
`DBMS_LDAP_UTL.INVALID_SUBSCRIBER_ORCL_CTX`	Invalid Subscriber Oracle Context.
`DBMS_LDAP_UTL.NO_SUCH_SUBSCRIBER`	Subscriber doesn't exist.
`DBMS_LDAP_UTL.MULTIPLE_SUBSCRIBER_ENTRIES`	The subscriber has multiple DN entries.
`DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX`	Invalid Root Oracle Context.
`DBMS_LDAP_UTL.ACCT_TOTALLY_LOCKED_EXCP`	User account is locked.

Table 11–6 (Cont.) AUTHENTICATE_USER Function Return Values

Value	Description
DBMS_LDAP_UTL.AUTH_PASSWD_CHANGE_WARN	This return value is deprecated.
DBMS_LDAP_UTL.AUTH_FAILURE_EXCP	Authentication failed.
DBMS_LDAP_UTL.PWD_EXPIRED_EXCP	User password has expired.
DBMS_LDAP_UTL.PWD_GRACELOGIN_WARN	Grace login for user.
DBMS_LDAP error codes	Return proper DBMS_LDAP error codes for unconditional failures that occurred when LDAP operations were carried out.

Usage Notes

This function can be called only after a valid LDAP session is obtained from a call to DBMS_LDAP.init().

See Also

DBMS_LDAP.init(), DBMS_LDAP_UTL.create_user_handle().

Function create_user_handle

The function create_user_handle() creates a user handle.

Syntax

```
FUNCTION create_user_handle
(
user_hd OUT HANDLE,
user_type IN PLS_INTEGER,
user_id IN VARCHAR2,
)
RETURN PLS_INTEGER;
```

Parameters

Table 11–7 CREATE_USER_HANDLE Function Parameters

Parameter Name	Parameter Type	Parameter Description
user_hd	HANDLE	A pointer to a handle to a user.
user_type	PLS_INTEGER	The type of user ID that is passed. Valid values for this argument are as follows: ■ DBMS_LDAP_UTL.TYPE_DN ■ DBMS_LDAP_UTL.TYPE_GUID ■ DBMS_LDAP_UTL.TYPE_NICKNAME
user_id	VARCHAR2	The user ID representing the user entry.

Return Values

Table 11–8 CREATE_USER_HANDLE Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.

Table 11–8 (Cont.) CREATE_USER_HANDLE Function Return Values

Value	Description
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.

See Also

DBMS_LDAP_UTL.get_user_properties(), DBMS_LDAP_UTL.set_user_handle_properties().

Function set_user_handle_properties

The function set_user_handle_properties() configures the user handle properties.

Syntax

```
FUNCTION set_user_handle_properties
(
user_hd IN HANDLE,
property_type IN PLS_INTEGER,
property IN HANDLE
)
RETURN PLS_INTEGER;
```

Parameters

Table 11–9 SET_USER_HANDLE_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
user_hd	HANDLE	A pointer to a handle to a user.
property_type	PLS_INTEGER	The type of property that is passed. Valid values for this argument are as follows: - DBMS_LDAP_UTL.SUBSCRIBER_HANDLE.
property	HANDLE	The property describing the user entry.

Return Values

Table 11–10 SET_USER_HANDLE_PROPERTIES Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.RESET_HANDLE	When a caller tries to reset the existing handle properties.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.

Usage Notes

The subscriber handle does not have to be set in User Handle Properties if the user handle is created with TYPE_DN or TYPE_GUID as the user type.

See Also

DBMS_LDAP_UTL.get_user_properties().

Function get_user_properties

The function get_user_properties() retrieves the user properties.

Syntax

```
FUNCTION get_user_properties
(
ld IN SESSION,
user_handle IN HANDLE,
attrs IN STRING_COLLECTION,
ptype IN PLS_INTEGER,
ret_pset_coll OUT PROPERTY_SET_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

Table 11–11 GET_USER_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
ld	SESSION	A valid LDAP session handle.
user_handle	HANDLE	The user handle.
attrs	STRING_COLLECTION	The list of user attributes to retrieve.
ptype	PLS_INTEGER	Type of properties to return. These are valid values: ■ DBMS_LDAP_UTL.ENTRY_PROPERTIES ■ DBMS_LDAP_UTL.NICKNAME_PROPERTY
ret-pset_collection	PROPERTY_SET_COLLECTION	User details contained in attributes requested by the caller.

Return Values

Table 11–12 GET_USER_PROPERTIES Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.NO_SUCH_USER	User does not exist.
DBMS_LDAP_UTL.MULTIPLE_USER_ENTRIES	The user has multiple DN entries.
DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX	Invalid root Oracle Context.

Table 11–12 (Cont.) GET_USER_PROPERTIES Function Return Values

Value	Description
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.
DBMS_LDAP error codes	Return proper DBMS_LDAP error codes for unconditional failures that occur when LDAP operations are carried out.

Usage Notes

This function requires the following:

- A valid LDAP session handle, which must be obtained from the DBMS_LDAP.init() function.
- A valid subscriber handle to be set in the group handle properties if the user type is of DBMS_LDAP_UTL.TYPE_NICKNAME.

This function does not identify a NULL subscriber handle as a default subscriber. The default subscriber can be obtained from DBMS_LDAP_UTL.create_subscriber_handle(), where a NULL subscriber_id is passed as an argument.

If the group type is either DBMS_LDAP_UTL.TYPE_GUID or DBMS_LDAP_UTL.TYPE_DN, the subscriber handle need not be set in the user handle properties. If the subscriber handle is set, it is ignored.

See Also

DBMS_LDAP.init(), DBMS_LDAP_UTL.create_user_handle().

Function set_user_properties

The function set_user_properties() modifies the properties of a user.

Syntax

```
FUNCTION set_user_properties
(
ld IN SESSION,
user_handle IN HANDLE,
pset_type IN PLS_INTEGER,
mod_pset IN PROPERTY_SET,
mod_op IN PLS_INTEGER
)
RETURN PLS_INTEGER;
```

Parameters

Table 11–13 SET_USER_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Description
ld	SESSION	A valid LDAP session handle.
user_handle	HANDLE	The user handle.
pset_type	PLS_INTEGER	The type of property set being modified. A valid value is ENTRY_PROPERTIES.
mod_pset	PROPERTY_SET	Data structure containing modify operations to perform on the property set.

Table 11–13 (Cont.) SET_USER_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Description
mod_op	PLS_INTEGER	The type of modify operation to be performed on the property set. Here are valid values: ■ ADD_PROPERTYSET ■ MODIFY_PROPERTYSET ■ DELETE_PROPERTYSET

Return Values

Table 11–14 SET_USER_PROPERTIES Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.NO_SUCH_USER	User does not exist.
DBMS_LDAP_UTL.MULTIPLE_USER_ENTRIES	The user has multiple DN entries.
DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX	Invalid root Oracle Context.
DBMS_LDAP_UTL.PWD_MIN_LENGTH_ERROR	Password length is less than the minimum required length.
DBMS_LDAP_UTL.PWD_NUMERIC_ERROR	Password must contain numeric characters.
DBMS_LDAP_UTL.PWD_NULL_ERROR	Password cannot be NULL.
DBMS_LDAP_UTL.PWD_INHISTORY_ERROR	Password cannot be the same as the one that is being replaced.
DBMS_LDAP_UTL.PWD_ILLEGALVALUE_ERROR	Password contains illegal characters.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.
DBMS_LDAP error codes	Return proper DBMS_LDAP error codes for unconditional failures while carrying out LDAP operations by the LDAP server.

Usage Notes

This function can only be called after a valid LDAP session is obtained from a call to DBMS_LDAP.init().

See Also

DBMS_LDAP.init(), DBMS_LDAP_UTL.get_user_properties().

Function get_user_extended_properties

The function get_user_extended_properties() retrieves user extended properties.

Syntax

```
FUNCTION get_user_extended_properties
(
ld IN SESSION,
user_handle IN HANDLE,
attrs IN STRING_COLLECTION
ptype IN PLS_INTEGER,
filter IN VARCHAR2,
rep_pset_coll OUT PROPERTY_SET_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

Table 11–15 GET_USER_EXTENDED_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
ld	SESSION	A valid LDAP session handle.
user_handle	HANDLE	The user handle.
attrs	STRING_COLLECTION	A list of attributes to fetch for the user.
ptype	PLS_INTEGER	The type of properties to return. Here is a valid value: - DBMS_LDAP_UTL.EXTPROPTYPE_RAD
filter	VARCHAR2	An LDAP filter to further refine the user properties returned by the function.
ret_pset_collection	PROPERTY_SET_COLLECTION	The user details containing the attributes requested by the caller.

Return Values

Table 11–16 GET_USER_EXTENDED_PROPERTIES Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.NO_SUCH_USER	User does not exist.
DBMS_LDAP_UTL.MULTIPLE_USER_ENTRIES	The user has multiple DN entries.
USER_PROPERTY_NOT_FOUND	User extended property does not exist.
DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX	Invalid root Oracle Context.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.
DBMS_LDAP error codes	Return proper DBMS_LDAP error codes for unconditional failures that occur when LDAP operations are carried out.

Usage Notes

This function can be called only after a valid LDAP session is obtained from a call to DBMS_LDAP.init().

See Also

DBMS_LDAP.init(), DBMS_LDAP_UTL.get_user_properties().

Function get_user_dn

The function get_user_dn() returns the user DN.

Syntax

```
FUNCTION get_user_dn
(
ld IN SESSION,
user_handle IN HANDLE,
dn OUT VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters

Table 11–17 GET_USER_DN Function Parameters

Parameter Name	Parameter Type	Parameter Description
ld	SESSION	A valid LDAP session handle.
user_handle	HANDLE	The user handle.
dn	VARCHAR2	The user DN.

Return Values

Table 11–18 GET_USER_DN Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.GENERAL_ERROR	Authentication failed.
DBMS_LDAP_UTL.NO_SUCH_USER	User does not exist.
DBMS_LDAP_UTL.MULTIPLE_USER_ENTRIES	The user has multiple DN entries.
DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX	Invalid root Oracle Context.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.
DBMS_LDAP error codes	Return proper DBMS_LDAP error codes for unconditional failures that occur when LDAP operations are carried out.

Usage Notes

This function can be called only after a valid LDAP session is obtained from a call to DBMS_LDAP.init().

See Also

DBMS_LDAP.init().

Function check_group_membership

The function check_group_membership() checks whether the user belongs to a group.

Syntax

```
FUNCTION check_group_membership
(
ld IN SESSION,
user_handle IN HANDLE,
group_handle IN HANDLE,
nested IN PLS_INTEGER
)
RETURN PLS_INTEGER;
```

Parameters

Table 11–19 CHECK_GROUP_MEMBERSHIP Function Parameters

Parameter Name	Parameter Type	Parameter Description
ld	SESSION	A valid LDAP session handle.
user_handle	HANDLE	The user handle.
group_handle	HANDLE	The group handle.
nested	PLS_INTEGER	The type of membership the user holds in groups. Here are valid values: ■ DBMS_LDAP_UTL.NESTED_MEMBERSHIP ■ DBMS_LDAP_UTL.DIRECT_MEMBERSHIP

Return Values

Table 11–20 CHECK_GROUP_MEMBERSHIP Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	If user is a member.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.GROUP_MEMBERSHIP	If user is not a member.

Usage Notes

This function can be called only after a valid LDAP session is obtained from a call to DBMS_LDAP.init().

See Also

DBMS_LDAP.get_group_membership().

Function locate_subscriber_for_user

The function locate_subscriber_for_user() retrieves the subscriber for the given user and returns a handle to it.

Syntax

```
FUNCTION locate_subscriber_for_user
(
ld IN SESSION,
user_handle IN HANDLE,
subscriber_handle OUT HANDLE
)
RETURN PLS_INTEGER;
```

Parameters

Table 11–21 LOCATE_SUBSCRIBER_FOR_USER Function Parameters

Parameter Name	Parameter Type	Parameter Description
ld	SESSION	A valid LDAP session handle.
user_handle	HANDLE	The user handle.
subscriber_handle	HANDLE	The subscriber handle.

Return Values

Table 11–22 LOCATE SUBSCRIBER FOR USER Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.NO_SUCH_SUBSCRIBER	Subscriber doesn't exist.
DBMS_LDAP_UTL.MULTIPLE_SUBSCRIBER_ENTRIES	Multiple number of subscriber DN entries exist in the directory for the given subscriber.
DBMS_LDAP_UTL.NO_SUCH_USER	User doesn’t exist.
DBMS_LDAP_UTL.MULTIPLE_USER_ENTRIES	Multiple number of user DN entries exist in the directory for the given user.
DBMS_LDAP_UTL.SUBSCRIBER_NOT_FOUND	Unable to locate subscriber for the given user.
DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX	Invalid Root Oracle Context.
DBMS_LDAP_UTL.ACCT_TOTALLY_LOCKED_EXCP	User account is locked.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.
DBMS_LDAP error codes	Return proper DBMS_LDAP error codes for unconditional failures while carrying out LDAP operations by the LDAP server.

Usage Notes

This function can be called only after a valid LDAP session is obtained from a call to DBMS_LDAP.init().

See Also

DBMS_LDAP.init(), DBMS_LDAP_UTL.create_user_handle().

Function get_group_membership

The function `get_group_membership()` returns the list of groups to which the user is a member.

Syntax

```
FUNCTION get_group_membership
(
user_handle IN HANDLE,
nested IN PLS_INTEGER,
attr_list IN STRING_COLLECTION,
ret_groups OUT PROPERTY_SET_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

Table 11–23 GET_GROUP_MEMBERSHIP Function Parameters

Parameter Name	Parameter Type	Parameter Description
`ld`	`SESSION`	A valid LDAP session handle.
`user_handle`	`HANDLE`	The user handle.
`nested`	`PLS_INTEGER`	The type of membership the user holds in groups. Here are valid values: ▪ `DBMS_LDAP_UTL.NESTED_MEMBERSHIP` ▪ `DBMS_LDAP_UTL.DIRECT_MEMBERSHIP`
`attr_list`	`STRING_COLLECTION`	A list of attributes to be returned.
`ret_groups`	`PROPERTY_SET_COLLECTION`	A pointer to a pointer to an array of group entries.

Return Values

Table 11–24 GET_GROUP_MEMBERSHIP Function Return Values

Value	Description
`DBMS_LDAP_UTL.SUCCESS`	On a successful completion.
`DBMS_LDAP_UTL.PARAM_ERROR`	Invalid input parameters.
`DBMS_LDAP_UTL.GENERAL_ERROR`	Other error.

Usage Notes

This function can be called only after a valid LDAP session is obtained from a call to `DBMS_LDAP.init()`.

See Also

`DBMS_LDAP.init()`.

Group-Related Subprograms

A group is represented using by using the DBMS_LDAP_UTL.HANDLE data type. A group handle represents a valid group entry. You can create a group handle by using a DN, GUID or a simple name, along with the appropriate subscriber handle. When a simple name is used, additional information from the Root Oracle Context and the Subscriber Oracle Context is used to identify the group. Here is an example of a group handle creation:

```
retval := DBMS_LDAP_UTL.create_group_handle(
group_handle,
DBMS_LDAP_UTL.TYPE_DN,
"cn=group1,cn=Groups,o=acme,dc=com"
);
```

This group handle has to be associated with an appropriate subscriber handle. For example, given a subscriber handle: *subscriber_handle* representing o=acme,dc=com, the subscriber handle can be associated in the following way:

```
retval := DBMS_LDAP_UTL.set_group_handle_properties(
group_handle,
DBMS_LDAP_UTL.SUBSCRIBER_HANDLE,
subscriber_handle
);
```

A sample use of group handle is getting group properties. Here is an example:

```
my_attrs is of type DBMS_LDAP.STRING_COLLECTION
my_attrs(1) := 'uniquemember';
retval := DBMS_LDAP_UTL.get_group_properties(
my_session,
my_attrs,
DBMS_LDAP_UTL.ENTRY_PROPERTIES,
my_pset_coll
);
```

The group-related subprograms also support membership-related functionality. Given a user handle, you can find out if it is a direct or a nested member of a group by using the DBMS_LDAP_UTL.check_group_membership() function. Here is an example:

```
retval := DBMS_LDAP_UTL.check_group_membership(
session,
user_handle,
group_handle,
DBMS_LDAP_UTL.DIRECT_MEMBERSHIP
```

You can also obtain a list of groups that a particular group belongs to, using the DBMS_LDAP_UTL.get_group_membership() function. For example:

```
my_attrs is of type DBMS_LDAP.STRING_COLLECTION
my_attrs(1) := 'cn';
retval := DBMS_LDAP_UTL.get_group_membership(
my_session,
user_handle,
DBMS_LDAP_UTL.DIRECT_MEMBERSHIP,
my_attrs
my_pset_coll
);
```

> **See Also:** Example: User-Related Functions on page B-9 for more examples of group handles

Function create_group_handle

The function `create_group_handle()` creates a group handle.

Syntax

```
FUNCTION create_group_handle
(
group_hd OUT HANDLE,
group_type IN PLS_INTEGER,
group_id IN VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters

Table 11–25 CREATE_GROUP_HANDLE Function Parameters

Parameter Name	Parameter Type	Parameter Description
`group_hd`	`HANDLE`	A pointer to a handle to a group.
`group_type`	`PLS_INTEGER`	The type of group ID that is passed. Valid values for this argument are as follows: ■ `DBMS_LDAP_UTL.TYPE_DN` ■ `DBMS_LDAP_UTL.TYPE_GUID` ■ `DBMS_LDAP_UTL.TYPE_NICKNAME`
`group_id`	`VARCHAR2`	The group ID representing the group entry.

Return Values

Table 11–26 CREATE_GROUP_HANDLE Function Return Values

Value	Description
`DBMS_LDAP_UTL.SUCCESS`	On a successful completion.
`DBMS_LDAP_UTL.PARAM_ERROR`	Invalid input parameters.
`DBMS_LDAP_UTL.GENERAL_ERROR`	Other error.

See Also

`DBMS_LDAP_UTL.get_group_properties()`, `DBMS_LDAP_UTL.set_group_handle_properties()`.

Function set_group_handle_properties

The function `set_group_handle_properties()` configures the group handle properties.

Syntax

```
FUNCTION set_group_handle_properties
(
group_hd IN HANDLE,
property_type IN PLS_INTEGER,
property IN HANDLE
)
RETURN PLS_INTEGER;
```

Parameters

Table 11–27 SET_GROUP_HANDLE_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
group_hd	HANDLE	A pointer to the handle to the group.
property_type	PLS_INTEGER	The type of property that is passed. Valid values for this argument are as follows: DBMS_LDAP_UTL.GROUP_HANDLE
property	HANDLE	The property describing the group entry.

Return Values

Table 11–28 SET_GROUP_HANDLE_PROPERTIES Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.RESET_HANDLE	When a caller tries to reset the existing handle properties.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.

Usage Notes

The subscriber handle doesn't need to be set in Group Handle Properties if the group handle is created with TYPE_DN or TYPE_GUID as the group type.

See Also

DBMS_LDAP_UTL.get_group_properties().

Function get_group_properties

The function get_group_properties() retrieves the group properties.

Syntax

```
FUNCTION get_group_properties
(
ld IN SESSION,
group_handle IN HANDLE,
attrs IN STRING_COLLECTION,
ptype IN PLS_INTEGER,
ret_pset_coll OUT PROPERTY_SET_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

Table 11–29 GET_GROUP_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
ld	SESSION	A valid LDAP session handle.
group_handle	HANDLE	The group handle.

Table 11–29 (Cont.) GET_GROUP_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
attrs	STRING_COLLECTION	A list of attributes that must be fetched for the group.
ptype	PLS_INTEGER	The type of properties to be returned. The valid value is DBMS_LDAP_UTL.ENTRY_PROPERTIES
ret_pset_coll	PROPERTY_SET_COLLECTION	The group details containing the attributes requested by the caller.

Return Values

Table 11–30 GET_GROUP_PROPERTIES Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.NO_SUCH_GROUP	Group doesn't exist.
DBMS_LDAP_UTL.MULTIPLE_GROUP_ENTRIES	Multiple number of group DN entries exist in the directory for the given group.
DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX	Invalid Root Oracle Context.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.
DBMS_LDAP error codes	Return proper DBMS_LDAP error codes for unconditional failures while carrying out LDAP operations by the LDAP server.

Usage Notes

This function requires the following:

- A valid LDAP session handle which must be obtained from the DBMS_LDAP.init() function.
- A valid subscriber handle to be set in the group handle properties if the group type is of: DBMS_LDAP_UTL.TYPE_NICKNAME.

This function doe not identify a NULL subscriber handle as a default subscriber. The default subscriber can be obtained fromDBMS_LDAP_UTL.create_subscriber_handle(), where a NULL subscriber_id is passed as an argument.

If the group type is either DBMS_LDAP_UTL.TYPE_GUID or DBMS_LDAP_UTL.TYPE_DN, the subscriber handle does not have to be set in the group handle properties. If the subscriber handle is set, it is ignored.

See Also

DBMS_LDAP.init(), DBMS_LDAP_UTL.create_group_handle().

Function get_group_dn

The function get_group_dn()returns the group DN.

Syntax

```
FUNCTION get_group_dn
(
ld IN SESSION,
group_handle IN HANDLE
dn OUT VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters

Table 11–31 GET_GROUP_DN Function Parameters

Parameter Name	Parameter Type	Parameter Description
ld	SESSION	A valid LDAP session handle.
group_handle	HANDLE	The group handle.
dn	VARCHAR2	The group DN.

Return Values

Table 11–32 GET_GROUP_DN Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.NO_SUCH_GROUP	Group doesn't exist.
DBMS_LDAP_UTL.MULTIPLE_GROUP_ENTRIES	Multiple number of group DN entries exist in the directory for the given group.
DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX	Invalid Root Oracle Context.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.
DBMS_LDAP error codes	Return proper DBMS_LDAP error codes for unconditional failures that are encountered when LDAP operations are carried out.

Usage Notes

This function can only be called after a valid LDAP session is obtained from a call to DBMS_LDAP.init().

See Also

DBMS_LDAP.init().

Subscriber-Related Subprograms

A subscriber is represented by using dbms_ldap_utl.handle data type. You can create a subscriber handle by using a DN, GUID or simple name. When a simple name is used, additional information from the root Oracle Context is used to identify the subscriber. This example shows a subscriber handle being created:

```
retval := DBMS_LDAP_UTL.create_subscriber_handle(
subscriber_handle,
DBMS_LDAP_UTL.TYPE_DN,
```

```
"o=acme,dc=com"
);
```

`subscriber_handle` is created by it's DN: `o=oracle,dc=com`.

Getting subscriber properties is one common use of a subscriber handle. Here is an example:

```
my_attrs is of type DBMS_LDAP.STRING_COLLECTION
       my_attrs(1) := 'orclguid';
       retval := DBMS_LDAP_UTL.get_subscriber_properties(
my_session,
my_attrs,
DBMS_LDAP_UTL.ENTRY_PROPERTIES,
my_pset_coll
);
```

> **See Also:** "DBMS_LDAP_UTL Sample Code" on page B-9 for examples of subscriber handles

Function create_subscriber_handle

The function `create_subscriber_handle()` creates a subscriber handle.

Syntax

```
FUNCTION create_subscriber_handle
(
subscriber_hd OUT HANDLE,
subscriber_type IN PLS_INTEGER,
subscriber_id IN VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters

Table 11–33 CREATE_SUBSCRIBER_HANDLE Function Parameters

Parameter Name	Parameter Type	Parameter Description
`subscriber_hd`	`HANDLE`	A pointer to a handle to a subscriber.
`subscriber_type`	`PLS_INTEGER`	The type of subscriber ID that is passed. Valid values for this argument are: ■ `DBMS_LDAP_UTL.TYPE_DN` ■ `DBMS_LDAP_UTL.TYPE_GUID` ■ `DBMS_LDAP_UTL.TYPE_NICKNAME` ■ `DBMS_LDAP_UTL.TYPE_DEFAULT`
`subscriber_id`	`VARCHAR2`	The subscriber ID representing the subscriber entry. This can be `NULL` if `subscriber_type` is `DBMS_LDAP_UTL.TYPE_DEFAULT`. In this case, the default subscriber is retrieved from the root Oracle Context.

Return Values

Table 11–34 CREATE_SUBSCRIBER_HANDLE Function Return Values

Value	Description
`DBMS_LDAP_UTL.SUCCESS`	On a successful completion.

Table 11–34 (Cont.) CREATE_SUBSCRIBER_HANDLE Function Return Values

Value	Description
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.

See Also

DBMS_LDAP_UTL.get_subscriber_properties().

Function get_subscriber_properties

The function get_subscriber_properties()retrieves the subscriber properties for the given subscriber handle.

Syntax

```
FUNCTION get_subscriber_properties
(
ld IN SESSION,
subscriber_handle IN HANDLE,
attrs IN STRING_COLLECTION,
ptype IN PLS_INTEGER,
ret_pset_coll OUT PROPERTY_SET_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

Table 11–35 GET_SUBSCRIBER_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
ld	SESSION	A valid LDAP session handle.
subscriber_handle	HANDLE	The subscriber handle.
attrs	STRING_COLLECTION	A list of attributes that must be retrieved for the subscriber.
ptype	PLS_INTEGER	Properties of the subscriber's Oracle Context to return. These are valid values: ▪ DBMS_LDAP_UTL.ENTRY_PROPERTIES ▪ DBMS_LDAP_UTL.COMMON_PROPERTIES
ret_pset_coll	PROPERTY_SET_COLLECTION	The subscriber details containing the attributes requested by the caller.

Return Values

Table 11–36 GET_SUBSCRIBER_PROPERTIES Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.NO_SUCH_SUBSCRIBER	Subscriber doesn't exist.

Table 11–36 (Cont.) GET_SUBSCRIBER_PROPERTIES Function Return Values

Value	Description
DBMS_LDAP_UTL.MULTIPLE_SUBSCRIBER_ENTRIES	Subscriber has a multiple number of DN entries.
DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX	Invalid root Oracle Context.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.
DBMS_LDAP error codes	Return proper DBMS_LDAP error codes for unconditional failures encountered while LDAP operations are carried out.

Usage Notes

This function can only be called after a valid LDAP session is obtained from a call to DBMS_LDAP.init().

See Also

DBMS_LDAP.init(), DBMS_LDAP_UTL.create_subscriber_handle().

Function get_subscriber_dn

The function get_subscriber_dn() returns the subscriber DN.

Syntax

```
FUNCTION get_subscriber_dn
(
ld IN SESSION,
subscriber_handle IN HANDLE,
dn OUT VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters

Table 11–37 GET_SUBSCRIBER_DN Function Parameters

Parameter Name	Parameter Type	Parameter Description
ld	SESSION	A valid LDAP session handle.
subscriber_handle	HANDLE	The subscriber handle.
dn	VARCHAR2	The subscriber DN.

Return Values

Table 11–38 GET_SUBSCRIBER_DN Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.NO_SUCH_SUBSCRIBER	Subscriber doesn't exist.
DBMS_LDAP_UTL.MULTIPLE_SUBSCRIBER_ENTRIES	Multiple number of subscriber DN entries exist in the directory for the given subscriber.

Table 11–38 (Cont.) GET_SUBSCRIBER_DN Function Return Values

Value	Description
DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX	Invalid root Oracle Context.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.
DBMS_LDAP error codes	Return proper DBMS_LDAP error codes for unconditional failures encountered when LDAP operations are carried out.

Usage Notes

This function can only be called after a valid LDAP session is obtained from a call to DBMS_LDAP.init().

See Also

DBMS_LDAP.init().

Function get_subscriber_ext_properties

The function get_subscriber_ext_properties() retrieves the subscriber extended properties. Currently this can be used to retrieve the subscriber-wide default Resource Access Descriptors.

Syntax

```
FUNCTION get_subscriber_ext_properties
(
ld IN SESSION,
subscriber_handle IN HANDLE,
attrs IN STRING_COLLECTION,
ptype IN PLS_INTEGER,
filter IN VARCHAR2,
rep_pset_coll OUT PROPERTY_SET_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

Table 11–39 GET_SUBSCRIBER_EXT_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
ld	SESSION	A valid LDAP session handle.
subscriber_handle	HANDLE	The subscriber handle.
attrs	STRING_COLLECTION	A list of subscriber attributes to retrieve.
ptype	PLS_INTEGER	The type of properties to return. A valid value is - DBMS_LDAP_UTL.DEFAULT_RAD_PROPERTIES
filter	VARCHAR2	An LDAP filter to further refine the subscriber properties returned by the function.

Table 11–39 (Cont.) GET_SUBSCRIBER_EXT_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
ret_pset_collection	PROPERTY_SET_COLLECTION	The subscriber details containing the attributes requested by the caller.

Return Values

Table 11–40 GET_USER_EXTENDED_PROPERTIES Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.NO_SUCH_USER	User does not exist.
DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX	Invalid root Oracle Context.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.
DBMS_LDAP error codes	Return proper DBMS_LDAP error codes for unconditional failures encountered when LDAP operations are carried out.

Usage Notes

This function can be called only after a valid LDAP session is obtained from a call to DBMS_LDAP.init().

See Also DBMS_LDAP.init(), DBMS_LDAP_UTL.get_subscriber_properties().

Property-Related Subprograms

Many of the user-related, subscriber-related, and group-related subprograms return DBMS_LDAP_UTL.PROPERTY_SET_COLLECTION, which is a collection of one or more LDAP entries representing results. Each of these entries is represented by a DBMS_LDAP_UTL.PROPERTY_SET. A PROPERTY_SET may contain attributes—that is, properties—and its values. Here is an example that illustrates the retrieval of properties from DBMS_LDAP_UTL.PROPERTY_SET_COLLECTION:

```
my_attrs is of type DBMS_LDAP.STRING_COLLECTION
my_attrs(1) := 'cn';

retval := DBMS_LDAP_UTL.get_group_membership(
my_session,
user_handle,
DBMS_LDAP_UTL.DIRECT_MEMBERSHIP,
my_attrs,
my_pset_coll
);

IF my_pset_coll.count > 0 THEN
      FOR i in my_pset_coll.first .. my_pset_coll.last LOOP
--    my_property_names is of type DBMS_LDAP.STRING_COLLECTION
       retval := DBMS_LDAP_UTL.get_property_names(
pset_coll(i),
property_names
       IF my_property_names.count > 0 THEN
           FOR j in my_property_names.first .. my_property_names.last LOOP
```

```
            retval := DBMS_LDAP_UTL.get_property_values(
pset_coll(i),
property_names(j),
property_values
            if my_property_values.COUNT > 0 then
             FOR k in my_property_values.FIRST..my_property_values.LAST LOOP
                    DBMS_OUTPUT.PUT_LINE(my_property_names(j)  ||  ': '
                                                                          ||my_
property_values(k));
                  END LOOP; -- For each value
            else
               DBMS_OUTPUT.PUT_LINE('NO VALUES FOR ' || my_property_names(j));
            end if;
          END LOOP; -- For each property name
        END IF; -- IF my_property_names.count > 0
     END LOOP; -- For each propertyset
  END IF; -- If my_pset_coll.count > 0
```

`use_handle` is a user handle. `my_pset_coll` contains all the nested groups that `user_handle` belongs to. The code loops through the resulting entries and prints out the `cn` of each entry.

> **See Also:** Example: User-Related Functions on page B-9 for more usage samples of the Property-related subpropgrams

Miscellaneous Subprograms

The miscellaneous subprograms in the `DBMS_LDAP_UTL` package perform a variety of different functions.

Function normalize_dn_with_case

The function `normalize_dn_with_case()` removes unnecessary white space characters from a DN and converts all characters to lower case based on a flag.

Syntax

```
FUNCTION normalize_dn_with_case
(
dn IN VARCHAR2,
lower_case IN PLS_INTEGER,
norm_dn OUT VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters

Table 11–41 NORMALIZE_DN_WITH_CASE Function Parameters

Parameter Name	Parameter Type	Parameter Description
`dn`	`VARCHAR2`	The DN.
`lower_case`	`PLS_INTEGER`	If set to `1`: The normalized DN returns in lower case. If set to `0`: The case is preserved in the normalized DN string.
`norm_dn`	`VARCHAR2`	The normalized DN.

Return Values

Table 11–42 NORMALIZE_DN_WITH_CASE Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.GENERAL_ERROR	On failure.

Usage Notes

This function can be used while comparing two DNs.

Function get_property_names

The function `get_property_names()` retrieves the list of property names in the property set.

Syntax

```
FUNCTION get_property_names
(
pset IN PROPERTY_SET,
property_names OUT STRING_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

Table 11–43 GET_PROPERTY_NAMES Function Parameters

Parameter Name	Parameter Type	Parameter Description
pset	PROPERTY_SET	The property set in the property set collection returned from any of the following functions: ■ DBMS_LDAP_UTL.get_group_membership() ■ DBMS_LDAP_UTL.get_subscriber_properties() ■ DBMS_LDAP_UTL.get_user_properties() ■ DBMS_LDAP_UTL.get_group_properties()
property_names	STRING_COLLECTION	A list of property names associated with the property set.

Return Values

Table 11–44 GET_PROPERTY_NAMES Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.GENERAL_ERROR	On error.

See Also

DBMS_LDAP_UTL.get_property values().

Function get_property_values

The function get_property_values() retrieves the property values (the strings) for a given property name and property.

Syntax

```
FUNCTION get_property_values
(
pset IN PROPERTY_SET,
property_name IN VARCHAR2,
property_values OUT STRING_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

Table 11–45 GET_PROPERTY_VALUES Function Parameters

Parameter Name	Parameter Type	Parameter Description
property_name	VARCHAR2	The property name.
pset	PROPERTY_SET	The property set in the property set collection obtained from any of the following function returns: ▪ DBMS_LDAP_UTL.get_group_membership() ▪ DBMS_LDAP_UTL.get_subscriber_properties() ▪ DBMS_LDAP_UTL.get_user_properties() ▪ DBMS_LDAP_UTL.get_group_properties()
property_values	STRING_COLLECTION	A list of property values (strings).

Return Values

Table 11–46 GET_PROPERTY_VALUES Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.GENERAL_ERROR	On failure.

See Also

DBMS_LDAP_UTL.get_property_values_len().

Function get_property_values_len

The function get_property_values_len() retrieves the binary property values for a given property name and property.

Syntax

```
FUNCTION get_property_values_len
(
pset IN PROPERTY_SET,
property_name IN VARCHAR2,
auth_type IN PLS_INTEGER,
property_values OUT BINVAL_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

Table 11–47 GET_PROPERTY_VALUES_LEN Function Parameters

Parameter Name	Parameter Type	Parameter Description
property_name	VARCHAR2	A property name.
pset	PROPERTY_SET	The property set in the property set collection obtained from any of the following function returns: ■ DBMS_LDAP_UTL.get_group_membership() ■ DBMS_LDAP_UTL.get_subscriber_properties() ■ DBMS_LDAP_UTL.get_user_properties() ■ DBMS_LDAP_UTL.get_group_properties()
property_values	BINVAL_COLLECTION	A list of binary property values.

Return Values

Table 11–48 GET_PROPERTY_VALUES_LEN Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.GENERAL_ERROR	On failure.

See Also

DBMS_LDAP_UTL.get_property_values().

Procedure free_propertyset_collection

The procedure free_propertyset_collection() frees the memory associated with property set collection.

Syntax

```
PROCEDURE free_propertyset_collection
(
pset_collection IN OUT PROPERTY_SET_COLLECTION
);
```

Parameters

Table 11–49 FREE_PROPERTYSET_COLLECTION Procedure Parameters

Parameter Name	Parameter Type	Parameter Description
pset_collection	PROPERTY_SET_COLLECTION	The property set collection returned from one of the following functions: ■ DBMS_LDAP_UTL.get_group_membership() ■ DBMS_LDAP_UTL.get_subscriber_properties() ■ DBMS_LDAP_UTL.get_user_properties() ■ DBMS_LDAP_UTL.get_group_properties()

See Also

DBMS_LDAP_UTL.get_group_membership(), DBMS_LDAP_UTL.get_subscriber_properties(), DBMS_LDAP_UTL.get_user_properties(), DBMS_LDAP_UTL.get_group_properties().

Function create_mod_propertyset

The function create_mod_propertyset() creates a MOD_PROPERTY_SET data structure.

Syntax

```
FUNCTION create_mod_propertyset
(
pset_type IN PLS_INTEGER,
pset_name IN VARCHAR2,
mod_pset OUT MOD_PROPERTY_SET
)
RETURN PLS_INTEGER;
```

Parameters

Table 11–50 CREATE_MOD_PROPERTYSET Function Parameters

Parameter Name	Parameter Type	Parameter Description
pset_type	PLS_INTEGER	The type of property set being modified. Here is a valid value: ENTRY_PROPERTIES
pset_name	VARCHAR2	The name of the property set. This can be NULL if ENTRY_PROPERTIES are being modified.
mod_pset	MOD_PROPERTY_SET	The data structure to contain modify operations to be performed on the property set.

Return Values

Table 11–51 CREATE_MOD_PROPERTYSET Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.

Table 11–51 (Cont.) CREATE_MOD_PROPERTYSET Function Return Values

Value	Description
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.

See Also

DBMS_LDAP_UTL.populate_mod_propertyset().

Function populate_mod_propertyset

The function populate_mod_propertyset() populates the MOD_PROPERTY_SET data structure.

Syntax

```
FUNCTION populate_mod_propertyset
(
mod_pset IN MOD_PROPERTY_SET,
property_mod_op IN PLS_INTEGER,
property_name IN VARCHAR2,
property_values IN STRING_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

Table 11–52 POPULATE_MOD_PROPERTYSET Function Parameters

Parameter Name	Parameter Type	Parameter Description
mod_pset	MOD_PROPERTY_SET	Mod-PropertySet data structure.
property_mod_op	PLS_INTEGER	The type of modify operation to perform on a property. These are valid values: ■ ADD_PROPERTY ■ REPLACE_PROPERTY ■ DELETE_PROPERTY
property_name	VARCHAR2	The name of the property
property_values	STRING_COLLECTION	Values associated with the property.

Return Values

Table 11–53 POPULATE_MOD_PROPERTYSET Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.GENERAL_ERROR	Authentication failed.
DBMS_LDAP_UTL.PWD_GRACELOGIN_WARN	Grace login for user.

See Also

DBMS_LDAP_UTL.create_mod_propertyset().

Procedure free_mod_propertyset

The procedure `free_mod_propertyset()` frees the `MOD_PROPERTY_SET` data structure.

Syntax

```
PROCEDURE free_mod_propertyset
(
mod_pset IN MOD_PROPERTY_SET
);
```

Parameters

Table 11–54 FREE_MOD_PROPERTYSET Procedure Parameters

Parameter Name	Parameter Type	Parameter Description
mod_pset	PROPERTY_SET	Mod_PropertySet data structure.

See Also

`DBMS_LDAP_UTL.create_mod_propertyset()`.

Procedure free_handle

The procedure `free_handle()` frees the memory associated with the handle.

Syntax

```
PROCEDURE free_handle
(
handle IN OUT HANDLE
);
```

Parameters

Table 11–55 FREE_HANDLE Procedure Parameters

Parameter Name	Parameter Type	Parameter Description
handle	HANDLE	A pointer to a handle.

See Also

`DBMS_LDAP_UTL.create_user_handle()`, `DBMS_LDAP_UTL.create_subscriber_handle()`, `DBMS_LDAP_UTL.create_group_handle()`.

Function check_interface_version

The function `check_interface_version()` checks the interface version.

Syntax

```
FUNCTION check_interface_version
(
interface_version IN VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters

Table 11–56 CHECK_INTERFACE_VERSION Function Parameters

Parameter Name	Parameter Type	Parameter Description
interface_version	VARCHAR2	Version of the interface.

Return Values

Table 11–57 CHECK_VERSION_INTERFACE Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	Interface version is supported.
DBMS_LDAP_UTL.GENERAL_ERROR	Interface version is not supported.

Function get_property_values_blob

The function `get_property_values_blob()` retrieves large binary property values for a given property name and property.

Syntax

```
FUNCTION get_property_values_blob
(
pset IN PROPERTY_SET,
property_name IN VARCHAR2,
auth_type IN PLS_INTEGER,
property_values OUT BLOB_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

Table 11–58 GET_PROPERTY_VALUES_BLOB Function Parameters

Parameters	Parameter Type	Description
property_name	VARCHAR2	A property name.
pset	PROPERTY_SET	The property set in the property set collection obtained from any of the following function returns: ■ DBMS_LDAP_UTL.get_group_membership() ■ DBMS_LDAP_UTL.get_subscriber_properties() ■ DBMS_LDAP_UTL.get_user_properties() ■ DBMS_LDAP_UTL.get_group_properties()
property_values	BLOB_COLLECTION	A list of binary property values.

Return Values

Table 11–59 GET_PROPERTY_VALUES_BLOB Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.GENERAL_ERROR	On failure.

See Also

DBMS_LDAP_UTL.get_property_values().

Procedure property_value_free_blob

Frees the memory associated with BLOB_COLLECTION returned by DBMS_LDAP.get_property_values_blob().

Syntax

```
Syntax
PROCEDURE property_value_free_blob
(
vals IN OUT DBMS_LDAP.BLOB_COLLECTION
);
```

Parameters

Table 11–60 PROPERTY_VALUE_FREE_BLOB Function Parameters

Parameter	Description
vals	The collection of large binary values returned by DBMS_LDAP.get_property_values_blob().

See Also

DBMS_LDAP.get_property_values_blob().

Function Return Code Summary

The DBMS_LDAP_UTL functions can return the values in the following table

Table 11–61 Function Return Codes

Name	Return Code	Description
SUCCESS	0	Operation successful.
GENERAL_ERROR	-1	This error code is returned on failure conditions other than those conditions listed here.
PARAM_ERROR	-2	Returned by all functions when an invalid input parameter is encountered.
NO_GROUP_MEMBERSHIP	-3	Returned by user-related functions and group functions when the user is not a member of a group.
NO_SUCH_SUBSCRIBER	-4	Returned by subscriber-related functions when the subscriber does not exist in the directory.

Table 11–61 (Cont.) Function Return Codes

Name	Return Code	Description
NO_SUCH_USER	-5	Returned by user-related functions when the user does not exist in the directory.
NO_ROOT_ORCL_CTX	-6	Returned by most functions when the root oracle context does not exist in the directory.
MULTIPLE_SUBSCRIBER_ ENTRIES	-7	Returned by subscriber-related functions when multiple subscriber entries are found for the given subscriber nickname.
INVALID_ROOT_ORCL_CTX	-8	Root Oracle Context does not contain all the required information needed by the function.
NO_SUBSCRIBER_ORCL_CTX	-9	Oracle Context does not exist for the subscriber.
INVALID_SUBSCRIBER_ ORCL_CTX	-10	Oracle Context for the subscriber is invalid.
MULTIPLE_USER_ENTRIES	-11	Returned by user-related functions when multiple user entries exist for the given user nickname.
NO_SUCH_GROUP	-12	Returned by group related functions when a group does not exist in the directory.
MULTIPLE_GROUP_ENTRIES	-13	Multiple group entries exist for the given group nickname in the directory.
ACCT_TOTALLY_LOCKED_ EXCEPTION	-14	Returned by `DBMS_LDAP_UTL.authenticate_user()` function when a user account is locked. This error is based on the password policy set in the subscriber oracle context.
AUTH_PASSWD_CHANGE_WARN	-15	This return code is deprecated.
AUTH_FAILURE_EXCEPTION	-16	Returned by `DBMS_LDAP_UTL.authenticate_user()` function when user authentication fails.
PWD_EXPIRED_EXCEPTION	-17	Returned by `DBMS_LDAP_UTL.authenticate_user()` function when the user password has expired. This is a password policy error.
RESET_HANDLE	-18	Returned when entity handle properties are being reset by the caller.
SUBSCRIBER_NOT_FOUND	-19	Returned by `DBMS_LDAP-UTL.locate_subscriber_for_user()` function when it is unable to locate the subscriber.
PWD_EXPIRE_WARN	-20	Returned by `DBMS_LDAP_UTL.authenticate_user()` function when the user password is about to expire. This is a password policy error.
PWD_MINLENGTH_ERROR	-21	Returned by `DBMS_LDAP_UTL.set_user_properties()` function while changing the user password and the new user password is less than the minimum required length. This is a password policy error.
PWD_NUMERIC_ERROR	-22	Returned by `DBMS_LDAP_UTL.set_user_properties()` function while changing the user password and the new user password does not contain at least one numeric character. This is a password policy error.

Table 11–61 (Cont.) Function Return Codes

Name	Return Code	Description
PWD_NULL_ERROR	-23	Returned by DBMS_LDAP_UTL.set_user_properties() function while changing the user password and the new user password is an empty password. This is a password policy error.
PWD_INHISTORY_ERROR	-24	Returned by DBMS_LDAP_UTL.set_user_properties() function while changing the user password and the new user password is the same as the previous password. This is a password policy error.
PWD_ILLEGALVALUE_ERROR	-25	Returned by DBMS_LDAP_UTL.set_user_properties() function while changing the user password and the new user password has an illegal character. This is a password policy error.
PWD_GRACELOGIN_WARN	-26	Returned by DBMS_LDAP_UTL.authenticate_user() function to indicate that the user password has expired and the user has been given a grace login. This is a password policy error.
PWD_MUSTCHANGE_ERROR	-27	Returned by DBMS_LDAP_UTL.authenticate_user() function when user password needs to be changed. This is a password policy error.
USER_ACCT_DISABLED_ERROR	-29	Returned by DBMS_LDAP_UTL.authenticate_user() function when user account has been disabled. This is a password policy error.
PROPERTY_NOT_FOUND	-30	Returned by user-related functions while searching for a user property in the directory.

Data Type Summary

The DBMS_LDAP_UTL package uses the data types in the following table

Table 11–62 DBMS_LDAP_UTL Data Types

Data Type	Purpose
HANDLE	Used to hold the entity.
PROPERTY_SET	Used to hold the properties of an entity.
PROPERTY_SET_COLLECTION	List of PROPERTY_SET structures.
MOD_PROPERTY_SET	Structure to hold modify operations on an entity.

12

DAS_URL Interface Reference

This chapter describes the Oracle extensions to the DAS_URL Service Interface. It contains these sections:

Directory Entries for the Service Units

Table 12–1 lists the Oracle Delegated Administration Services units and the directory entries that store relative URLs for these units.

Table 12–1 Service Units and Corresponding Entries

Service Unit	Entry
Create User	`cn=CreateUser,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext`
Edit User	`cn=EditUser,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext`
Edit User when GUID is passed as a parameter	`cn=EditUserGivenGUID,cn=OperationURLs,cn=DAS,cn=Products, cn=OracleContext`
Delete User	`cn=DeleteUser,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext`
Delete User when GUID of the user to be deleted is passed as a parameter	`cn=DeleteUserGivenGUID,cn=OperationURLs,cn=DAS,cn=Products, cn=OracleContext`
Create Group	`cn=CreateGroup,cn=OperationURLs,cn=DAS,cn=Products, cn=OracleContext`
Edit Group	`cn=EditGroup,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext`
Edit the group whose GUID is passed through a parameter	`cn=EditGroupGivenGUID,cn=OperationURLs,cn=DAS,cn=Products, cn=OracleContext`
Delete Group	`cn=DeleteGroup,cn=OperationURLs,cn=DAS,cn=Products, cn=OracleContext`
Delete group with the GUID passed through a parameter	`cn=DeleteGroupGivenGUID,cn=OperationURLs,cn=DAS,cn=Products, cn=OracleContext`

Table 12–1 (Cont.) Service Units and Corresponding Entries

Service Unit	Entry
Assign privileges to a user	`cn=UserPrivilege,cn=OperationURLs,cn=DAS,cn=Products, cn=OracleContext`
Assign privileges to a user with the GUID passed through a parameter	`cn=UserPrivilegeGivenGUID,cn=OperationURLs,cn=DAS,cn=Products, cn=OracleContext`
Assign privilege to a group	`cn=GroupPrivilege,cn=OperationURLs,cn=DAS,cn=Products, cn=OracleContext`
Assign privilege to a group with the given GUID	`cn=GroupPrivilegeGivenGUID,cn=OperationURLs,cn=DAS,cn=Products, cn=OracleContext`
View User account information/Profile	`cn=AccountInfo,cn=OperationURLs,cn=DAS,cn=Products, cn=OracleContext`
Edit User account Information/Profile	`cn=Edit My Profile,cn=OperationURLs,cn=DAS,cn=Products, cn=OracleContext`
Change Password	`cn=PasswordChange,cn=OperationURLs,cn=DAS,cn=Products, cn=OracleContext`
Search User	`cn=UserSearch,cn=OperationURLs,cn=DAS,cn=Products, cn=OracleContext`
Search Group	`cn=GroupSearch,cn=OperationURLs,cn=DAS,cn=Products, cn=OracleContext`
Search User LOV	`cn=UserLOV,cn=OperationURLs,cn=DAS,cn=Products, cn=OracleContext`
Search Group LOV	`cn=GroupLOV,cn=OperationURLs,cn=DAS,cn=Products, cn=OracleContext`
EUS Console	`cn=EUS Console,cn=OperationURLs,cn=DAS,cn=Products,cn=OracleContext"`
Delegation Console	`cn=DelegationConsole,cn=OperationURLs,cn=DAS,cn=Products, cn=OracleContext`

DAS Units and Corresponding URL Parameters

Table 12–2 lists the DAS units and the URL parameters that can be passed to these units.

Table 12–2 DAS Units and Corresponding URL Parameters

DAS Unit	Parameter	Return Values

Create User	`doneURL`
	`homeURL`
	`cancelURL`
	`enablePA`
		`returnGUID`

Edit User	`homeURL`
	`doneURL`
	`cancelURL`
	`enablePA`
		-

Table 12–2 (Cont.) DAS Units and Corresponding URL Parameters

DAS Unit	Parameter	Return Values
EditUserGivenGUID	homeURL doneURL cancelURL enablePA userGUID	-
EditMyProfile	homeURL doneURL cancelURL	-
Delegation Console	-	-
DeleteUser	homeURL doneURL cancelURL	-
DeleteUserGivenGUID	homeURL doneURL cancelURL userGUID	-
UserPrivilege	homeURL doneURL cancelURL	
UserPrivilegeGivenGUID	homeURL doneURL cancelURL userGUID	-
CreateGroup	homeURL doneURL cancelURL enablePA parentDN	returnGUID
EditGroup	homeURL doneURL cancelURL enablePA	-
EditGroupGivenGUID	homeURL doneURL cancelURL enablePA groupGUID	-
DeleteGroup	homeURL doneURL cancelURL	-
DeleteGroupGivenGUID	homeURL doneURL cancelURL groupGUID	-
GroupPrivilege	homeURL doneURL cancelURL	-

Table 12–2 (Cont.) DAS Units and Corresponding URL Parameters

DAS Unit	Parameter	Return Values
GroupPrivilegeGivenGUID	homeURL doneURL cancelURL groupGUID	-
AccountInfo	homeURL doneURL cancelURL	-
PasswordChange	homeURL doneURL cancelURL	-
UserSearch	homeURL doneURLm cancelURL	-
GroupSearch	homeURL doneURL cancelURL	-
UserLOV	base cfilter title dasdomain callbackURL	userDn userGuid userName nickName userEmail
GroupLOV	otype base cfilter title dasdomain callbackURL	groupDN groupGuid groupName groupDescription

DAS URL API Parameter Descriptions

The parameters described in Table 12–3 are used with DAS units.

Table 12–3 DAS URL Parameter Descriptions

Parameter	Description
homeURL	The URL that is linked to the global button **Home**. When the calling application specifies this value, clicking **Home** redirects the DAS unit to the URL specified by this parameter.
doneURL	This URL is used by DAS to redirect the DAS page at the end of each operation. In the case of Create User, once the user is created, clicking **OK** redirects the URL to this location.
callbackURL	DAS uses this URL to send return values to the invoking application. For UserLOV and GroupLOV units, the return values are submitted as HTML form parameters through the HTTP POST method.
cancelURL	This URL is linked with all the **Cancel** buttons shown in the DAS units. Any time the user clicks **Cancel**, the page is redirected to the URL specified by this parameter.
enablePA	This parameter takes a Boolean value of true or false. Set to true, the parameter enables the Assign Privileges in User or Group operation. If the enablePA is passed with value of true in the Create User page, the Assign Privileges to User section also appears in the Create User page.

Table 12–3 (Cont.) DAS URL Parameter Descriptions

Parameter	Description
`userGUID`	This is the GUID of the user to be edited or deleted. This corresponds to the orclguid attribute. Specifying the GUID causes the search for the user step in either editUser or deleteUser units to be skipped.
`GroupGUID`	This is the GUID of the group to be edited or deleted. This corresponds to the orclguid attribute. Specifying the GUID causes the search for the group step in either editGroup or deleteGroup units to be skipped.
`parentDN`	When this parameter is specified in CreateGroup, the group is created under this container. If the parameter is not specified, group creation defaults to the group search base.
`base`	This parameter represents the search base in the case of search operations.
`cfilter`	This parameter represents the filter to be used for the search. This filter is LDAP compliant.
`title`	This parameter represents the title to be shown in the Search and Select LOV page.
`otype`	This parameter represents the object type used for search. Values supported are `Select`, `Edit`, and `Assign`.
`returnGUID`	This parameter is appended to the done URL in case of a create operation. The value will be the orclguid of the new object.
`dasdomain`	This parameter is needed only when the browser is Internet Explorer and the calling URL and the DAS URL are on different hosts and in the same domain. An example value is us.oracle.com. Note the calling application also needs to set the `document.domain` parameter on the formload. For more details, refer to Microsoft support at: `http://support.microsoft.com/`

Search-and-Select Service Units for Users or Groups

DAS provides service units for searching and selecting users or groups. These service units are sometimes referred to as user or group List Of Values (LOV).

Invoking Search-and-Select Service Units for Users or Groups

A custom application can open a popup window and populate its contents by supplying a search-and-select URL for a user or group:

```
http://a.b.c:port/oiddas/ui/oracle/ldap/das/search/LOVUserSearch?title=User&
callbackurl=http://x.y.z:port/custapp/Callback
```

or

```
http://a.b.c:port/oiddas/ui/oracle/ldap/das/search/LOVGroupSearch?title=
Group&callbackurl=http://x.y.z:port/custapp/Callback
```

In these examples, `a.b.c:port` is the host name and port of the OID DAS application server. `x.y.z:port` is the host name and port of the custom application server. `title` is a string that appears in the title of the Search and Select page. `callbackurl` is a URL on the custom application server that receives the selected parameters for users or groups.

Note: To avoid popup blocking, the custom application may open the popup window with a URL on the local custom application server and immediately redirect to the OID DAS User or Group Search-and-Select URL.

Receiving Data from the User or Group Search-and-Select Service Units

After a User or Group has been selected via the OID DAS User or Group Search-and-Select Service Unit, an HTTP form will be submitted to the callbackurl page using the POST method. The parameters defined in Table 12–4 and Table 12–5 are available to the callbackurl page:

Table 12–4 User Search and Select

Parameter	Description
`userDn`	User's distinguished name.
`userGuid`	User's global unique ID.
`userName`	User's name.
`nickName`	User's nickname
`userEmail`	User's email.

Table 12–5 Group Search and Select

Parameter	Description
`groupDN`	Group's distinguished name.
`groupGuid`	Group's global unique ID.
`groupName`	Group's name.
`groupDescription`	Group's description.

The callbackurl page in the popup window may transfer the form parameters to the invoking page in the opener window using JavaScript. It may then close the popup window.

Note: To avoid JavaScript security problems, the custom application may supply the callbackurl page on the same server as the invoking page. This enables the callbackurl page in the popup window and the invoking page in the opener window to communicate directly through JavaScript.

13

Provisioning Integration API Reference

This chapter examines the registration API for the Oracle Directory Provisioning Integration Service. It contains the following sections:

Versioning of Provisioning Files and Interfaces

In release 9.0.2, the default interface version was version 1.1. In releases 9.0.4 and 10.1.2, the interface version defaults to version 2.0, but the administrator can set this back to version 1.1 to maintain the release 9.0.2 interface.

Extensible Event Definition Configuration

This feature is only for outbound events. It addresses the ability to define a new event at run time so that the provisioning integration service can interpret a change in Oracle Internet Directory and determine whether an appropriate event is to be generated and propagated to an application. The following events will be the only configured events at installation time.

An event definition (entry) consists of the following attributes.

- Event object type (`orclODIPProvEventObjectType`): This specifies the type of object the event is associated with. For example, the object could be a `USER`, `GROUP`, or `IDENTITY`.
- LDAP change type (`orclODIPProvEventChangeType`): This indicates that all kinds of LDAP operations can generate an event for this type of object. (e.g `ADD`, `MODIFY`, `DELETE`)
- Event criteria (`orclODIPProvEventCriteria`): The additional selection criteria that qualify an LDAP entry to be of a specific object type. For example, `Objectclass=orclUserV2` means that any LDAP entry that satisfies this criteria can be qualified as this Object Type and any change to this entry can generate appropriate events.

The object class that holds the above attributes is `orclODIPProvEventTypeConfig`. The container `cn=ProvisioningEventTypeConfig,cn=odi,cn=oracle internet directory` is used to store all the event type configurations.

Table 13–1 lists the event definitions predefined as a part of the installation.

Table 13–1 Predefined Event Definitions

Event Object Type	LDAP Change Type	Event Criteria
ENTRY	ADD MODIFY DELETE	objectclass=*
USER	ADD MODIFY DELETE	objectclass=interorgperson objectclass=orcluserv2
IDENTITY	ADD MODIFY DELETE	objectclass=interorgperson objectclass=orcluserv2
GROUP	ADD MODIFY DELETE	objectclass=orclgroup objectclass=groupofuniquenames
SUBSCRPTION	ADD MODIFY DELETE	objectclass=orclservicerecepient
SUBSCRIBER	ADD MODIFY DELETE	objectclass=orclsubscriber

The container `cn=ProvisioningEventTypeConfig,cn=odi,cn=oracle internet directory` is used to store all the event definition configurations. LDAP configuration of the predefined event definitions is as follows:

```
dn: orclODIPProvEventObjectType=ENTRY,cn=ProvisioningEventTypeConfig,cn=odi,
cn=oracle internet directory
orclODIPProvEventObjectType: ENTRY
orclODIPProvEventLDAPChangeType: Add
orclODIPProvEventLDAPChangeType: Modify
orclODIPProvEventLDAPChangeType: Delete
orclODIPProvEventCriteria: objectclass=*
objectclass: orclODIPProvEventTypeConfig

dn:
orclODIPProvEventObjectType=USER,cn=ProvisioningEventTypeConfig,cn=odi,cn=oracle
internet directory
orclODIPProvEventObjectType: USER
orclODIPProvEventLDAPChangeType: Add
orclODIPProvEventLDAPChangeType: Modify
orclODIPProvEventLDAPChangeType: Delete
orclODIPProvEventCriteria: objectclass=InetOrgPerson
orclODIPProvEventCriteria: objectclass=orcluserv2
objectclass: orclODIPProvEventTypeConfig

dn: orclODIPProvEventObjectType=IDENTITY,cn=ProvisioningEventTypeConfig,cn=odi,
cn=oracle internet directory
orclODIPProvEventObjectType: IDENTITY
orclODIPProvEventLDAPChangeType: Add
orclODIPProvEventLDAPChangeType: Modify
orclODIPProvEventLDAPChangeType: Delete
orclODIPProvEventCriteria: objectclass=inetorgperson
orclODIPProvEventCriteria: objectclass=orcluserv2
objectclass: orclODIPProvEventTypeConfig
```

```
dn: orclODIPProvEventObjectType=GROUP,cn=ProvisioningEventTypeConfig,cn=odi,
cn=oracle internet directory
orclODIPProvEventObjectType: GROUP
orclODIPProvEventLDAPChangeType: Add
orclODIPProvEventLDAPChangeType: Modify
orclODIPProvEventLDAPChangeType: Delete
orclODIPProvEventCriteria: objectclass=orclgroup
orclODIPProvEventCriteria: objectclass=groupofuniquenames
objectclass: orclODIPProvEventTypeConfig

dn:
orclODIPProvEventObjectType=SUBSCRIPTION,cn=ProvisioningEventTypeConfig,cn=odi,
cn=oracle internet directory
orclODIPProvEventObjectType: SUBSCRIPTION
orclODIPProvEventLDAPChangeType: Add
orclODIPProvEventLDAPChangeType: Modify
orclODIPProvEventLDAPChangeType: Delete
orclODIPProvEventCriteria: objectclass=orclservicerecepient
objectclass: orclODIPProvEventTypeConfig

dn: orclODIPProvEventObjectType=SUBSCRIBER,cn=ProvisioningEventTypeConfig,cn=odi,
cn=oracle internet directory
orclODIPProvEventObjectType: SUBSCRIBER
orclODIPProvEventLDAPChangeType: Add
orclODIPProvEventLDAPChangeType: Modify
orclODIPProvEventLDAPChangeType: Delete
orclODIPProvEventCriteria: objectclass=orclsubscriber
objectclass: orclODIPProvEventTypeConfig
```

To define a new event of Object type XYZ (which is qualified with the object class `objXYZ`), create the following entry in Oracle Internet Directory. The DIP server recognizes this new event definition and propagates events if necessary to applications that subscribe to this event.

```
dn: orclODIPProvEventObjectType=XYZ,cn=ProvisioningEventTypeConfig,cn=odi,
cn=oracle internet directory
orclODIPProvEventObjectType: XYZ
orclODIPProvEventLDAPChangeType: Add
orclODIPProvEventLDAPChangeType: Modify
orclODIPProvEventLDAPChangeType: Delete
orclODIPProvEventCriteria: objectclass=objXYZ
objectclass: orclODIPProvEventTypeConfig
```

This means that if an LDAP entry with the object class `objXYZ` is added, modified, or deleted, DIP will propagate the `XYZ_ADD`, `XYZ_MODIFY`, or `XYZ_DELETE` event to any application concerned.

Inbound and Outbound Events

An application can register as a supplier as well as a consumer of events. The provisioning subscription profile has the attributes described in Table 13–2 on page 13-4.

Table 13–2 Attributes of the Provisioning Subscription Profile

Attribute	Description
EventSubscriptions	Outbound events only (multivalued). Events for which DIP should send notification to this application. The format of this string is [USER]GROUP]:[*domain_of_interest*]:[DELETE|ADD|MODIFY(*list_of_attributes_separated_by_comma*)] Multiple values may be specified by listing the string multiple times, each time with different values. If parameters are not specified, the following defaults are assumed: USER:*organization_DN*:DELETEGROUP:*organization_DN*:DELETE—that is, send user and group delete notifications under the organization DN.
MappingRules	Inbound events Only (multivalued). This attribute is used to map the type of object received from an application and a qualifying filter condition to determine the domain of interest for this event. The mapping takes this form: *OBJECT_TYPE*: *Filter_condition*: *domain_of_interest* Multiple rules are allowed. In the mapping EMP:cn=users,dc=acme,dc=com, the object type received is EMP. The event is meant for the domain cn=users,dc=acme,dc=com. In the mapping EMP:l=AMERICA:l=AMER,cn=users,dc=acme,dc=com, the object type received is EMP. The event is meant for the domain l=AMER,cn=users,dc=acme,dc=com.
permittedOperations	Inbound events only (multi valued). This attribute is used to define the types of events an application is privileged to send to the provisioning integration service. The mapping takes this form: *Event_Object*: *affected_domain*:*operation*(*attributes*, . . .) In the mapping IDENTITY:cn=users,dc=acme,dc=com:ADD(*) the IDENTITY_ADD event is allowed for the specified domain and all attributes are also allowed. In the mapping IDENTITY:cn=users,dc=acme,dc=com:MODIFY(cn,sn.mail,telephonenumber), the IDENTITY_MODIFY event is allowed only for the attributes in the list. Any extra attributes are silently ignored.

PL/SQL Bidirectional Interface (Version 2.0)

The PL/SQL callback interface requires that you develop a PL/SQL package that the provisioning integration service invokes in the application-specific database. Choose any name for the package, but be sure to use the same name when you register the package at subscription time. Implement the package using the following PL/SQL package specification:

```
DROP TYPE LDAP_EVENT;
DROP TYPE LDAP_EVENT_STATUS;
DROP TYPE LDAP_ATTR_LIST;
DROP TYPE LDAP_ATTR;
--------------------------------------------------------------------------------
-- Name: LDAP_ATTR
-- Data Type: OBJECT

DESCRIPTION: This structure contains details regarding an attribute. A list of one
--           or more of this object is passed in any event.
--------------------------------------------------------------------------------
----------------
CREATE TYPE LDAP_ATTR AS OBJECT (
     attr_name        VARCHAR2(256),
```

```
     attr_value       VARCHAR2(4000),
     attr_bvalue      RAW(2048),
     attr_value_len   INTEGER,
     attr_type        INTEGER ,
     attr_mod_op      INTEGER
);

GRANT EXECUTE ON LDAP_ATTR to public;

CREATE TYPE LDAP_ATTR_LIST AS TABLE OF LDAP_ATTR;
/
GRANT EXECUTE ON LDAP_ATTR_LIST to public;

--------------------------------------------------------------------------------
-----------------
-- Name: LDAP_EVENT
-- Data Type: OBJECT
-- DESCRIPTION: This structure contains event information plus the attribute
--              list.
--------------------------------------------------------------------------------
-----------------

CREATE TYPE LDAP_EVENT AS OBJECT (
          event_type  VARCHAR2(32),
          event_id    VARCHAR2(32),
          event_src   VARCHAR2(1024),
          event_time  VARCHAR2(32),
          object_name VARCHAR2(1024),
          object_type VARCHAR2(32),
          object_guid VARCHAR2(32),
          object_dn   VARCHAR2(1024),
          profile_id  VARCHAR2(1024),
          attr_list   LDAP_ATTR_LIST ) ;
/

GRANT EXECUTE ON LDAP_EVENT to public;

--------------------------------------------------------------------------------
-----------------
-- Name: LDAP_EVENT_STATUS
-- Data Type: OBJECT
-- DESCRIPTION: This structure contains information that is sent by the
--              consumer of an event to the supplier in response to the
--              actual event.

--------------------------------------------------------------------------------
-----------------

CREATE TYPE LDAP_EVENT_STATUS AS OBJECT (
          event_id          VARCHAR2(32),
          orclguid          VARCHAR(32),
          error_code        INTEGER,
          error_String      VARCHAR2(1024),
          error_disposition VARCHAR2(32)) ;
/

GRANT EXECUTE ON LDAP_EVENT_STATUS to public;
```

Provisioning Event Interface (Version 1.1)

As stated in "Development Tasks for Provisioning Integration" on page 4-14, you must develop logic to consume events generated by the provisioning integration service.

The interface between the application and the provisioning integration service can be table-based, or it can use PL/SQL callbacks.

The PL/SQL callback interface requires that you develop a PL/SQL package that the provisioning integration service invokes in the application-specific database. Choose any name for the package, but be sure to use the same name when you register the package at subscription time. Implement the package using the following PL/SQL package specification:

```
Rem
Rem      NAME
Rem         ldap_ntfy.pks - Provisioning Notification Package Specification.
Rem

DROP TYPE LDAP_ATTR_LIST;
DROP TYPE LDAP_ATTR;

-- LDAP ATTR
-----------------------------------------------------------------
--
--  Name        : LDAP_ATTR
--  Data Type   : OBJECT
--  DESCRIPTION : This structure contains details regarding
--                an attribute.
--
-----------------------------------------------------------------
CREATE TYPE LDAP_ATTR AS OBJECT (
     attr_name        VARCHAR2(255),
     attr_value       VARCHAR2(2048),
     attr_bvalue      RAW(2048),
     attr_value_len   INTEGER,
     attr_type        INTEGER  -- (0 - String, 1 - Binary)
     attr_mod_op      INTEGER
);
/
 GRANT EXECUTE ON LDAP_ATTR to public;

-------------------------------------------------------------
--
--  Name        : LDAP_ATTR_LIST
--  Data Type   : COLLECTION
--  DESCRIPTION : This structure contains collection
--                of attributes.
--
-------------------------------------------------------------
CREATE TYPE LDAP_ATTR_LIST AS TABLE OF LDAP_ATTR;
/
 GRANT EXECUTE ON LDAP_ATTR_LIST to public;

-----------------------------------------------------------------------------
--
--  NAME        : LDAP_NTFY
--  DESCRIPTION : This is a notifier interface implemented by Provisioning System
--                clients to receive information about changes in Oracle Internet
--                Directory. The name of package can be customized as needed.
--                The function names within this package should not be changed.
```

```
--
--
-------------------------------------------------------------------------------
CREATE OR REPLACE PACKAGE LDAP_NTFY AS

--
-- LDAP_NTFY data type definitions
--


-- Event Types
USER_DELETE               CONSTANT VARCHAR2(256) := 'USER_DELETE';
USER_MODIFY               CONSTANT VARCHAR2(256) := 'USER_MODIFY';
GROUP_DELETE              CONSTANT VARCHAR2(256) := 'GROUP_DELETE';
GROUP_MODIFY              CONSTANT VARCHAR2(256) := 'GROUP_MODIFY';

-- Return Codes (Boolean)
SUCCESS                   CONSTANT NUMBER  := 1;
FAILURE                   CONSTANT NUMBER  := 0;

-- Values for attr_mod_op in LDAP_ATTR object.
MOD_ADD                   CONSTANT NUMBER  := 0;
MOD_DELETE                CONSTANT NUMBER  := 1;
MOD_REPLACE               CONSTANT NUMBER  := 2;
-------------------------------------------------------------------------------
-----------------
-- Name: LDAP_NTFY
-- DESCRIPTION: This is the interface to be implemented by Provisioning System
--              clients to send information to and receive information from
--              Oracle Internet Directory. The name of the package can be
--              customized as needed. The function names within this package
--              should not be changed.

-------------------------------------------------------------------------------
-----------------

CREATE OR REPLACE PACKAGE LDAP_NTFY AS
```

Predefined Event Types

```
ENTRY_ADD               CONSTANT VARCHAR2 (32)    := 'ENTRY_ADD';
ENTRY_DELETE            CONSTANT VARCHAR2 (32)    := 'ENTRY_DELETE';
ENTRY_MODIFY            CONSTANT VARCHAR2 (32)    := 'ENTRY_MODIFY';

USER_ADD                CONSTANT VARCHAR2 (32)    := 'USER_ADD';
USER_DELETE             CONSTANT VARCHAR2 (32)    := 'USER_DELETE';
USER_MODIFY CONSTANT    VARCHAR2(32)              := 'USER_MODIFY';

IDENTITY_ADD            CONSTANT VARCHAR2 (32)    := 'IDENTITY_ADD';
IDENTITY_DELETE         CONSTANT VARCHAR2 (32)    := 'IDENTITY_DELETE';
IDENTITY_MODIFY         CONSTANT VARCHAR2 (32)    := 'IDENTITY_MODIFY';

GROUP_ADD               CONSTANT VARCHAR2 (32)    := 'GROUP_ADD';
GROUP_DELETE            CONSTANT VARCHAR2 (32)    := 'GROUP_DELETE';
GROUP_MODIFY            CONSTANT VARCHAR2 (32)    := 'GROUP_MODIFY';

SUBSCRIPTION_ADD        CONSTANT VARCHAR2(32)     := 'SUBSCRIPTION_ADD';
SUBSCRIPTION_DELETE     CONSTANT VARCHAR2(32)     := 'SUBSCRIPTION_DELETE';
SUBSCRIPTION_MODI       CONSTANT VARCHAR2(32)     := 'SUBSCRIPTION_MODIFY';
```

```
SUBSCRIBER_ADD           CONSTANT VARCHAR2(32)     := 'SUBSCRIBER_ADD';
SUBSCRIBER_DELETE        CONSTANT VARCHAR2(32)     := 'SUBSCRIBER_DELETE';
SUBSCRIBER_MODIFY        CONSTANT VARCHAR2(32)     := 'SUBSCRIBER_MODIFY';
```

Attribute Type

```
ATTR_TYPE_STRING              CONSTANT NUMBER    := 0;
ATTR_TYPE_BINARY              CONSTANT NUMBER    := 1;
ATTR_TYPE_ENCRYPTED_STRING    CONSTANT NUMBER    := 2;
```

Attribute Modification Type

```
MOD_ADD        CONSTANT NUMBER    := 0;
MOD_DELETE     CONSTANT NUMBER    := 1;
MOD_REPLACE    CONSTANT NUMBER    := 2;
```

Event Dispositions Constants

```
EVENT_SUCCESS    CONSTANT VARCHAR2(32)    := 'EVENT_SUCCESS';
EVENT_ERROR      CONSTANT VARCHAR2(32)    := 'EVENT_ERROR';
EVENT_RESEND     CONSTANT VARCHAR2(32)    := 'EVENT_RESEND';
```

Callbacks

A callback is a function invoked by the provisioning integration service to send or receive notification events. While transferring events for an object, the related attributes can also be sent along with other details. The attributes are delivered as a collection (array) of attribute containers, which are in unnormalized form: if an attribute has two values, two rows are sent in the collection.

GetAppEvent()

The directory integration and provisioning server invokes this API in the remote database. It is up to the application to respond with an event. The Oracle Directory Integration and Provisioning platform processes the event and sends the status back using the `PutAppEventStatus()` callback. The return value of `GetAppEvent()` indicates whether an event is returned or not.

```
FUNCTION GetAppEvent (event OUT LDAP_EVENT)
RETURN NUMBER;

-- Return CONSTANTS
EVENT_FOUND          CONSTANT NUMBER  := 0;
EVENT_NOT_FOUND      CONSTANT NUMBER  := 1403;
```

If the provisioning server is not able to process the event—that is, it runs into some type of LDAP error—it responds with `EVENT_RESEND`. The application is expected to resend that event when `GetAppEvent()` is invoked again.

If the provisioning server is able to process the event, but finds that the event cannot be processed—for example, the user to be modified does not exist, or the user to be subscribed does not exist, or the user to be deleted does not exist—then it responds with `EVENT_ERROR` to indicate to the application that something was wrong. Resending the event is not required. It is up to the application to handle the event.

Note the difference between `EVENT_RESEND` and `EVENT_ERROR` in the previous discussion. `EVENT_RESEND` means that it was possible to apply the event but the server could not. If it gets the event again, it might succeed.

EVENT_ERROR means there is no error in performing directory operations, but the event could not be processed due to other reasons.

PutAppEventStatus()

The directory integration and provisioning server invokes this callback in the remote database after processing an event it has received using the GetAppEvent() callback. For every event received, the directory integration and provisioning server sends the status event back after processing the event.

```
PROCEDURE PutAppEventStatus (event_status IN LDAP_EVENT_STATUS);
```

PutOIDEvent()

The directory integration and provisioning server invokes this API in the remote database. It sends event to applications using this callback. It also expects a status event object in response as an OUT parameter. If a valid event status object is not sent back, or it indicates a RESEND, the directory integration and provisioning server resends the event. In case of EVENT_ERROR, the server does not resend the event.

```
PROCEDURE PutOIDEvent (event  IN  LDAP_EVENT,   event_status  OUT LDAP_EVENT_
STATUS);
END LDAP_NTFY;
/
```

Part III

Appendixes

Part III presents the command-line tools, including generic tools and Oracle-specific tools. It contains these appendixes:

- Appendix A, "Syntax for LDIF and Command-Line Tools"
- Appendix B, "DSML Syntax"

A

Syntax for LDIF and Command-Line Tools

This appendix provides syntax, usage notes, and examples for LDIF and LDAP command-line tools. It contains these topics:

- LDAP Data Interchange Format (LDIF) Syntax
- Starting, Stopping, Restarting, and Monitoring Oracle Internet Directory Servers
- Entry and Attribute Management Command-Line Tools Syntax
- Oracle Directory Integration and Provisioning Platform Command-Line Tools Syntax

LDAP Data Interchange Format (LDIF) Syntax

The standardized file format for directory entries is as follows:

```
dn: distinguished_name
attribute_type: attribute_value
.
.
.
objectClass: object_class_value
.
.
.
```

Property	Value	Description
`dn:`	*`RDN,RDN,RDN,...`*	Separate RDNs with commas.
`attribute_type`	*`attribute_value`*	This line repeats for every attribute in the entry, and for every attribute value in multi-valued attributes.
`objectClass`	*`object_class_value`*	This line repeats for every object class.

The following example shows a file entry for an employee. The first line contains the DN. The lines that follow the DN begin with the mnemonic for an attribute, followed by the value to be associated with that attribute. Note that each entry ends with lines defining the object classes for the entry.

```
dn: cn=Suzie Smith,ou=Server Technology,o=Acme, c=US
cn: Suzie Smith
cn: SuzieS
sn: Smith
mail: ssmith@us.Acme.com
telephoneNumber: 69332
```

```
photo: /ORACLE_HOME/empdir/photog/ssmith.jpg
objectClass: organizationalPerson
objectClass: person
objectClass: top
```

The next example shows a file entry for an organization:

```
dn: o=Acme,c=US
o: Acme
ou: Financial Applications
objectClass: organization
objectClass: top
```

LDIF Formatting Notes

A list of formatting rules follows. This list is not exhaustive.

- All mandatory attributes belonging to an entry being added must be included with non-null values in the LDIF file.

  Tip: To see the mandatory and optional attribute types for an object class, use Oracle Directory Manager. See *Oracle Internet Directory Administrator's Guide*.

- Non-printing characters and tabs are represented in attribute values by base-64 encoding.
- The entries in your file must be separated from each other by a blank line.
- A file must contain at least one entry.
- Lines can be continued to the next line by beginning the continuation line with a space or a tab.
- Add a blank line between separate entries.
- Reference binary files, such as photographs, with the absolute address of the file, preceded by two forward slashes.
- The DN contains the full, unique directory address for the object.
- The lines listed after the DN contain both the attributes and their values. DNs and attributes used in the input file must match the existing structure of the DIT. Do not use attributes in the input file that you have not implemented in your DIT.
- Sequence the entries in an LDIF file so that the DIT is created from the top down. If an entry relies on an earlier entry for its DN, make sure that the earlier entry is added before its child entry.
- When you define schema within an LDIF file, insert a white space between the opening parenthesis and the beginning of the text, and between the end of the text and the ending parenthesis.

  See Also:

  - The various resources listed in "Related Documents" on page xxv for a complete list of LDIF formatting rules
  - The section "Using Globalization Support with LDIF Files" in *Oracle Internet Directory Administrator's Guide*

Starting, Stopping, Restarting, and Monitoring Oracle Internet Directory Servers

This section tells how to use command-line tools for starting, stopping, restarting, and monitoring Oracle Internet Directory servers. It contains these topics:

The OID Monitor (oidmon) Syntax

Use the OID Monitor to initiate, monitor, and terminate directory server processes. If you elect to install a replication server, OID Monitor controls it. When you use `oidctl` to start or stop directory server instances, OID Monitor interprets your commands.

Starting the OID Monitor

Starting OID Monitor restarts any Oracle Internet Directory processes that were previously stopped.

To start the OID Monitor:

1. Set the following environment variables:
   - *ORACLE_HOME*
   - ORACLE_*SID* or a proper TNS CONNECT string
   - NLS_LANG (*APPROPRIATE_LANGUAGE*.AL32UTF8). The default language set at installation is AMERICAN_AMERICA.
   - PATH. In the PATH environment variable, specify the Oracle LDAP binary—that is, *ORACLE_HOME*/bin—before the UNIX binary directory.
2. At the system prompt, type:

```
oidmon [connect=connect_string] [host=virtual/host_name][sleep=seconds] start
```

Table A–1 Arguments for Starting OID Monitor

Argument	Description
connect=*connect_string*	Specifies the connect string for the database to which you want to connect. This is the network service name set in the tnsnames.ora file. This argument is optional.
host=*virtual/host_name*	Specifies the virtual host or rack nodes on which to start OID Monitor
sleep=*seconds*	Specifies number of seconds after which the OID Monitor should check for new requests from OID Control and for requests to restart any servers that may have stopped. The default sleep time is 10 seconds. This argument is optional.
start	Starts the OID Monitor process

For example:

```
oidmon connect=dbs1 sleep=15 start
```

To start OID Monitor on a virtual host:

```
oidmon connect=dbsl host=virtual_host start
```

Stopping the OID Monitor

Stopping the OID Monitor also stops all other Oracle Internet Directory processes.

To stop the OID Monitor daemon, at the system prompt, type:

```
oidmon [connect=connect_string] [host=virtual/host_name] stop
```

Table A–2 Arguments for Stopping OID Monitor

Argument	Description
`connect=connect_string`	Specifies the connect string for the database to which you want to connect. This is the connect string set in the `tnsnames.ora` file.
`host=virtual/host_name`	Specifies the virtual host or rack nodes on which to start OID Monitor
`stop`	Stops the OID Monitor process

For example:

```
oidmon connect=dbs1 stop
```

Starting and Stopping OID Monitor in a Cold Failover Cluster Configuration

While starting and stopping OID Monitor, use the `host` parameter to specify the virtual host name. The syntax is:

```
oidmon [connect=connect_string] host=virtual_host start|stop
```

> **Note:** If you are going to start Oracle Internet Directory servers on a virtual host, then, when using both `oidmon` and `oidctl`, be sure to specify the `host` argument as the virtual host.
>
> If the OID Monitor is started with the `host=host name` argument, and the host name does not match the name of the physical host, then the OID Monitor assumes that the intended host is the logical host. You must use the same host name when using `oidmon` to stop or start any servers, otherwise the OID Monitor does not start or stop the servers.
>
> To determine the physical host name, execute the `uname` command.

The OID Control Utility (oidctl) Syntax

OID Control Utility is a command-line tool for starting and stopping the directory server. The commands are interpreted and executed by the OID Monitor process.

> **Note:** Although you can start the directory server without using OID Monitor and the OID Control Utility, Oracle Corporation recommends that you use them. This way, if the directory server unexpectedly terminates, then OID Monitor automatically restarts it.

This section contains these topics:

- Starting and Stopping an Oracle Directory Server Instance
- Troubleshooting Directory Server Instance Startup
- Starting and Stopping an Oracle Directory Replication Server Instance

- Starting the Oracle Directory Integration and Provisioning Server
- Stopping the Oracle Directory Integration and Provisioning Server
- Restarting Oracle Internet Directory Server Instances
- Starting and Stopping Directory Servers on a Virtual Host or an Oracle Application Server Cluster (Identity Management)

Starting and Stopping an Oracle Directory Server Instance

Use the object class to start and stop Oracle directory server instances.

Starting an Oracle Directory Server Instance The syntax for starting an Oracle directory server instance is:

```
oidctl connect=connect_string server=oidldapd instance=server_instance_number
-server number_of_server_processes [configset=configset_number]
[host=virtual/host_name] [flags=' -p port_number -work
maximum_number_of_worker_threads_per_server -debug debug_level
-l change_logging'] start
```

Table A–3 Arguments for Starting a Directory Server by Using OIDCTL

Argument	Description
`-debug debug_level`	Specifies a debug level during Oracle directory server instance startup
`-l change_logging`	Turns replication change logging on and off. To turn it off, enter `-l false`. To turn it on, do any one of the following: ■ omit the `-l` flag ■ enter simply `-l` ■ enter `-l true` Turning off change logging for a given node by specifying `-l false` has two drawbacks: it prevents replication of updates on that node to other nodes in the DRG, and it prevents application provisioning and synchronization of connected directories, because those two services require an active change log. The default, `TRUE`, permits replication, provisioning, and synchronization.
`-p port_number`	Specifies a port number during server instance startup. The default port number is `389`.
`-server server_processes`	Specifies the number of server processes to start on this port
`-sport`	Specifies the SSL port number during server instance startup. Default port if not set is `636`. **See Also:** ■ The information about `orclsslenable` attribute in the section "Configuration Set Entry Schema Elements" in *Oracle Internet Directory Administrator's Guide* ■ "Configuring SSL Parameters"in *Oracle Internet Directory Administrator's Guide*
`-work threads_per_server`	Specifies the maximum number of worker threads for this server

Table A–3 (Cont.) Arguments for Starting a Directory Server by Using OIDCTL

Argument	Description
`configset=`*`configset_number`*	Configset number used to start the server. This defaults to `configset0` if not set. This should be a number between 0 and 1000.
`connect=`*`connect_string`*	If you already have a `tnsnames.ora` file configured, then this is the net service name specified in that file, located in *`ORACLE_HOME`*`/network/admin`.
`host=`*`virtual/host_name`*	Specifies the virtual host or rack nodes on which to start the directory server
`instance=`*`instance_number`*	Instance number of the server to start. Should be a number between 1 and 1000.
`server=oidldapd`	Type of server to start (valid values are `OIDLDAPD` and `OIDREPLD`). This is not case-sensitive.
`start`	Starts the server specified in the `server` argument.

For example, to start a directory server instance whose net service name is `dbs1`, using `configset5` at port `12000`, with a debug level of `1024`, an instance number `3`, and in which change logging is turned off, type at the system prompt:

```
oidctl connect=dbs1 server=oidldapd instance=3 configset=5 flags='-p 12000
-debug 1024 -l ' start
```

When starting and stopping an Oracle directory server instance, the server name and instance number are mandatory, as are the commands `start` or `stop`. All other arguments are optional.

All keyword value pairs within the flags arguments must be separated by a single space.

Single quotes are mandatory around the flags.

The configset identifier defaults to zero (`configset0`) if not set.

> **Note:** If you choose to use a port other than the default port (`389` for non-secure usage or `636` for secure usage), you must tell the clients which port to use to locate the Oracle Internet Directory. If you use the default ports, clients can connect to the Oracle Internet Directory without referencing a port in their connect requests.

Stopping an Oracle Directory Server Instance At the system prompt, type:

```
oidctl connect=connect_string server=oidldapd instance=server_instance_number stop
```

For example:

```
oidctl connect=dbs1 server=oidldapd instance=3 stop
```

Troubleshooting Directory Server Instance Startup

If the directory server fails to start, you can override all user-specified configuration parameters to start the directory server and then return the configuration sets to a workable state by using the `ldapmodify` operation.

To start the directory server by using its hard-coded default parameters instead of the configuration parameters stored in the directory, type at the system prompt:

```
oidctl connect=connect_string flags='-p port_number -f'
```

The `-f` option in the flags starts the server with hard-coded configuration values, overriding any defined configuration sets except for the values in `configset0`.

To see debug log files generated by the OID Control Utility, navigate to *`ORACLE_HOME`*`/ldap/log`.

Starting and Stopping an Oracle Directory Replication Server Instance

Use the OID Control Utility to start and stop Oracle directory replication server instances.

Starting an Oracle Directory Replication Server Instance The syntax for starting the Oracle directory replication server is:

```
oidctl connect=connect_string server=oidrepld instance=server_instance_number
[configset=configset_number] flags= ' -p directory_server_port_number
-d debug_level -h directory_server_host_name -m [true | false]
-z transaction_size ' start
```

Table A–4 Arguments for Starting a Directory Replication Server by Using OIDCTL

Argument	Description
`connect=`*`connect_string`*	If you already have a `tnsnames.ora` file configured, then this is the name specified in that file, which is located in *`ORACLE_HOME`*`/network/admin`.
`server=oidrepld`	Type of server to start (valid values are `OIDLDAPD` and `OIDREPLD`). This is not case-sensitive.
`instance=`*`instance_number`*	Instance number of the server to start. Should be a number between 1 and 1000.
`configset=`*`configset_number`*	Configset number used to start the server. The default is `configset0`. This should be a number between 0 and 1000.
`-p` *`directory__port`*	Port number that the replication server uses to connect to the directory on TCP port *`directory_server_port_number`*. If you do not specify this option, the tool connects to the default port (`389`).
`-d` *`debug_level`*	Specifies a debug level during replication server instance startup
`-h` *`directory_host_name`*	Specifies the *`directory_server_host_name`* to which the replication server connects, rather than to the default host, that is, your local computer. *`directory_server_host_name`* can be a computer name or an IP address. (Replication server only)
`-m [true|false]`	Turns conflict resolution on and off. Valid values are `true` and `false`. The default is true. (Replication server only)
`-z` *`transaction_size`*	Specifies the number of changes applied in each replication update cycle. If you do not specify this, the number is determined by the Oracle directory server sizelimit parameter, which has a default setting of `1024`. You can configure this latter setting.
`start`	Starts the server specified in the *`server`* argument.

For example, to start the replication server with an `instance=1`, at port `12000`, with debugging set to `1024`, type at the system prompt:

```
oidctl connect=dbs1 server=oidrepld instance=1 flags='-p 12000 -h eastsun11 -d
1024' start
```

When starting and stopping an Oracle directory replication server, the `-h` flag, which specifies the host name, is mandatory. All other flags are optional.

All keyword value pairs within the flags arguments must be separated by a single space.

Single quotes are mandatory around the flags.

The configset identifier defaults to zero (`configset0`) if not set.

> **Note:** If you choose to use a port other than the default port (`389` for non-secure usage or `636` for secure usage), you must tell the clients which port to use to locate the Oracle Internet Directory. If you use the default ports, clients can connect to the Oracle Internet Directory without referencing a port in their connect requests.

Stopping an Oracle Directory Replication Server Instance At the system prompt, type:

```
oidctl connect=connect_string server=OIDREPLD instance=server_instance_number stop
```

For example:

```
oidctl connect=dbs1 server=oidrepld instance=1 stop
```

Starting the Oracle Directory Integration and Provisioning Server

The Oracle directory integration and provisioning server executable, `odisrv`, resides in the *`ORACLE_HOME`*`/bin` directory.

The way you start the directory integration and provisioning server depends on whether your installation is:

- A typical Oracle Internet Directory installation

  In this case, your installation includes, among other server and client components, the OID Monitor and the OID Control Utility. In such installations, you start and stop the directory integration and provisioning server by using these tools.

  > **Note:** Although you can start the directory integration and provisioning server without using the OID Monitor and the OID Control Utility, Oracle Corporation recommends that you use them. This way, if the directory integration and provisioning server unexpectedly terminates, the OID Monitor automatically restarts it.

- An Oracle Directory Integration and Provisioning platform-only installation

  In this case, the way you start the directory integration and provisioning server depends on whether you are using the Oracle Directory Integration and Provisioning platform for high availability.

  – If you are using Oracle Directory Integration and Provisioning platform for high availability, then Oracle Corporation recommends that you start the directory integration and provisioning server by using the OID Monitor and the OID Control Utility. This requires configuring the `tnsnames.ora` file with the right host and SID to which the OID Monitor must connect.

– If you are not using Oracle Directory Integration and Provisioning platform for high availability, then Oracle Corporation recommends that you start the directory integration and provisioning server without using the OID Monitor.

You can start the directory integration and provisioning server in either SSL mode for tighter security, or non-SSL mode. You need to use a connect string to connect to the database.

> **Note:** When the Oracle directory integration and provisioning server is invoked in the default mode, it supports only the Oracle Directory Provisioning Integration Service, and not the Oracle Directory Synchronization Service.

Starting the Oracle Directory Integration and Provisioning Server by Using the OID Monitor and Control Utilities

To start the directory integration and provisioning server in non-SSL mode:

1. Be sure that OID Monitor is running. To verify this on UNIX, enter the following at the command line:

   ```
   ps -ef | grep oidmon
   ```

   If OID Monitor is not running, then start it by following the instructions in "The OID Monitor (oidmon) Syntax" on page A-3.

2. Start the directory integration and provisioning server by using the OID Control Utility. Do this by entering:

   ```
   oidctl [connect=connect_string] server=odisrv [instance=instance_number]
   [config=configuration_set_number] [flags="[host=hostname] [port=port_number]
   [grpID=group_identifier_of_provisioning_profile] [debug=debug_level]
   [refresh=interval_between_refresh] [maxprofiles=number_of_profiles]
   [sslauth=ssl_mode ]"] start
   ```

   Table A–5 describes the arguments in this command.

Table A–5 Description of Arguments for Starting the Oracle Directory Integration and Provisioning Server

Argument	Description
`connect=`*`connect_string`*	If you already have a `tnsnames.ora` file configured, then this is the net service name specified in that file, located in *`ORACLE_HOME`*`/network/admin`
`server=odisrv`	Type of server to start. In this case, the server you are starting is `odisrv`. This is not case-sensitive. This argument is mandatory.
`instance=`*`instance_number`*	Specifies the instance number to assign to the directory integration and provisioning server. This instance number must be unique. OID Monitor verifies that the instance number is not already associated with a currently running instance of this server. If it is associated with a currently running instance, then OID Monitor returns an error message.
`config=`*`set_number`*	Specifies the number of the configuration set that the directory integration and provisioning server is to execute. This argument is mandatory.
`host=`*`host_name`*	Oracle directory server host name

Table A–5 (Cont.) Description of Arguments for Starting the Oracle Directory Integration and Provisioning Server

Argument	Description
`port=port_number`	Oracle directory server port number
`debug=debug_level`	The required debugging level of the directory integration and provisioning server. See the chapter about logging, auditing, and monitoring in *Oracle Internet Directory Administrator's Guide*.
`refresh=refresh_interval`	Specifies the interval, in minutes, between server refreshes for any changes in the integration profiles. The default is 2 minutes (`Refresh=2`).
`grpID=profile_identifier`	Specifies the group of profiles to be scheduled.
`maxprofiles=number_of_profiles`	Specifies the maximum number of profiles that can be executed concurrently for this server instance.

`sslauth=ssl_mode`	SSL modes:
	■ `0`: SSL is not used—that is, non-SSL mode
	■ `1`: SSL used for encryption only—that is, with no PKI authentication. A wallet is not used in this case.
	■ `2`: SSL is used with one-way authentication. This mode requires you to specify a complete path name of an Oracle Wallet, including the file name itself, unlike other directory tools that expect only the wallet location. For example, in a server-only installation, or in a complete installation, you would enter something like this:
	`oidctl server=odisrv [instance=instance_number] [configset=configset_number] [grpID=group_identifier_of_provisioning_profile] flags="host=myhost port=myport sslauth=2`
	In a client-only installation, you would enter something like this:
	`odisrv [host=host_name] [port=port_number] config=configuration_set_number [instance=instance_number] [debug=debug_level] [refresh=interval_between_refresh] [maxprofiles=number_of_profiles] [refresh=interval_between_refresh] [maxprofiles=number_of_profiles] [sslauth=ssl_mode]`

Starting the Oracle Directory Integration and Provisioning Server Without Using the OID Monitor and the OID Control Utility In a client-only installation, where the OID Monitor and OID Control tools are not available, the Oracle directory integration and provisioning server can be started without OID Monitor or OID Control Utility, either in non-SSL mode or, for tighter security, in SSL mode. The parameters described in Table A–5 on page A-9 remain the parameters for each type of invocation.

To start the directory integration and provisioning server, enter the following at the command line:

```
odisrv [host=host_name] [port=port_number] config=configuration_set_number
[instance=instance_number] [debug=debug_level] [refresh=interval_between_refresh]
[maxprofiles=number_of_profiles] [sslauth=ssl_mode]
```

Stopping the Oracle Directory Integration and Provisioning Server

You can use the OID Monitor and the OID Control utility to stop the directory integration and provisioning server:

1. Before you stop the directory integration and provisioning server, be sure that the OID Monitor is running. To verify this, enter the following at the command line:

   ```
   ps -ef | grep oidmon
   ```

   If OID Monitor is not running, then start it by following the instructions in "The OID Monitor (oidmon) Syntax" on page A-3.

2. Stop the directory integration and provisioning server by entering:

   ```
   oidctl [connect=connect_string] server=odisrv instance=instance stop
   ```

Note: To run shell script tools on the Windows operating system, you need one of the following UNIX emulation utilities:

- Cygwin 1.3.2.2-1 or later. Visit this site:

  `http://sources.redhat.com`

- MKS Toolkit 6.1. Visit this site:

  `http://www.datafocus.com/`

Restarting Oracle Internet Directory Server Instances

When you want to refresh the server cache immediately, rather than at the next scheduled time, use the `RESTART` command. When the Oracle Internet Directory server restarts, it maintains the same parameters it had before it stopped.

To restart an Oracle Internet Directory server instance, at the system prompt, type:

```
oidctl connect=connect_string server={oidldapd|oidrepld|odisrv}
instance=server_instance_number  restart
```

OID Monitor must be running whenever you restart directory server instances.

If you try to contact a server that is not running, you receive from the SDK the error message `81—LDAP_SERVER_DOWN`.

If you change a configuration set entry that is referenced by an active server instance, you must stop that instance and restart it to effect the changed value in the configuration set entry on that server instance. You can either issue the `STOP` command followed by the `START` command, or you can use the `RESTART` command. `RESTART` both stops and restarts the server instance.

For example, suppose that Oracle directory server `instance1` is started, using `configset3`, and with the net service name `dbs1`. Further, suppose that, while `instance1` is running, you change one of the attributes in `configset3`. To enable the change in `configset3` to take effect on `instance1`, you enter the following command:

```
oidctl connect=dbs1 server=oidldapd instance=1 restart
```

If there are more than one instance of the Oracle directory server running on that node using `configset3`, then you can restart all the instances at once by using the following command syntax:

```
oidctl connect=dbs1 server=oidldapd restart
```

Note that this command restarts all the instances running on the node, whether they are using `configset3` or not.

Important Note: During the restart process, clients cannot access the Oracle directory server instance. However, the process takes only a few seconds to execute.

Starting and Stopping Directory Servers on a Virtual Host or an Oracle Application Server Cluster (Identity Management)

When starting a directory server, a directory replication server, or a directory integration and provisioning server, use the `host` parameter to specify the virtual host name.

Starting and Stopping a Directory Server

To start a directory server on a virtual host:

```
oidctl [connect=connect_string] host=virtual_host_name server=oidldapd
instance=instance_number configset=configset_number flags= "..." start
```

To stop a directory server on a virtual host:

```
oidctl host=virtual_host_name server=oidldapd instance=instance_number stop
```

Starting and Stopping a Directory Replication Server

To start a directory replication server on a virtual host:

```
oidctl [connect=connect_string] host=virtual_host_name server=oidrepld
instance=instance_number flags= "..." start
```

To stop a directory replication server on a virtual host:

```
oidctl host=virtual_host_name server=oidrepld instance=instance_number stop
```

Starting and Stopping a Directory Integration and Provisioning Server

To start a directory integration and provisioning server on a virtual host:

```
oidctl [connect=connect_string] host=virtual_host_name server=odisrv
instance=instance_number configset=configset_number flags= "..." start
```

To stop a directory integration and provisioning server on a virtual host:

```
oidctl host=virtual/host_name server=odisrv instance=instance_number stop
```

When the directory server is started to run on the virtual host, it binds and listens to requests on the specified LDAP port on the IP address or IP addresses that correspond to the virtual host only.

When communicating with the directory server, the directory replication server uses the virtual host name. Further, the `replicaID` attribute that represents the unique replication identification for the Oracle Internet Directory node is generated once. It is

independent of the host name and hence requires no special treatment in an Oracle Application Server Cold Failover Cluster.

When communicating with the directory server, the directory integration and provisioning server uses the virtual host name.

Entry and Attribute Management Command-Line Tools Syntax

This section tells you how to use the following tools:

- The Catalog Management Tool (catalog.sh) Syntax
- ldapadd Syntax
- ldapbind Syntax
- ldapcompare Syntax
- ldapdelete Syntax
- ldapmoddn Syntax
- ldapmodify Syntax
- ldapmodifymt Syntax
- ldapsearch Syntax

Note: Various UNIX shells interpret some characters—for example, asterisks (*)—as special characters. Depending on the shell you are using, you may need to escape these characters.

The Catalog Management Tool (catalog.sh) Syntax

Oracle Internet Directory uses indexes to make attributes available for searches. When Oracle Internet Directory is installed, the `cn=catalogs` entry lists available attributes that can be used in a search. You can index only those attributes that have:

- An equality matching rule
- Matching rules supported by Oracle Internet Directory

If you want to use additional attributes in search filters, then you must add them to the catalog entry. You can do this at the time you create the attribute by using Oracle Directory Manager. However, if the attribute already exists, then you can index it only by using the Catalog Management tool.

Before running `catalog.sh`, be sure that the directory server is either stopped or in read-only mode. Otherwise, data will be inconsistent.

Caution: Do not use the `catalog.sh -delete` option on indexes created by the Oracle Internet Directory base schema. Removing indexes from base schema attributes can adversely impact the operation of Oracle Internet Directory.

Note: To run shell script tools on the Windows operating system, you need one of the following UNIX emulation utilities:

- Cygwin 1.3.2.2-1 or later. Visit this site:

  `http://sources.redhat.com`

- MKS Toolkit 6.1. Visit this site:

  `http://www.datafocus.com`

The Catalog Management tool uses this syntax:

```
catalog.sh -connect connect_string {-add|-delete} {-attr attr_name|-file file_
name}
```

Table A–6 Arguments for the Catalog Management Tool (catalog.sh)

Argument	Description
`-connect connect_string`	Specifies the connect string to connect to the directory database. This argument is mandatory. **See Also:** *Oracle9i Net Services Administrator's Guide* in the Oracle Database Documentation Library
`-add -attr attr_name`	Indexes the specified attribute
`-delete -attr attr_name`	Drops the index from the specified attribute
`-add -file file_name`	Indexes attributes (one for each line) in the specified file
`-delete -file file_name`	Drops the indexes from the attributes in the specified file

When you enter the `catalog.sh` command, the following message appears:

```
This tool can only be executed if you know the OiD user password.
Enter OiD password:
```

If you enter the correct password, the command is executed. If you give an incorrect password, the following message is displayed:

```
Cannot execute this tool
```

To effect the changes after running the Catalog Management tool, stop, then restart, the Oracle directory server.

See Also:

- "The OID Control Utility (oidctl) Syntax" on page A-4 and for instructions on starting and restarting directory servers. Note that OID Monitor must be running before you start a directory server.
- "The OID Monitor (oidmon) Syntax" on page A-3 for information about starting OID Monitor
- The section about matching rules in the schema appendix of *Oracle Internet Directory Administrator's Guide*

ldapadd Syntax

The `ldapadd` command-line tool enables you to add entries, their object classes, attributes, and values to the directory. To add attributes to an existing entry, use the `ldapmodify` command, explained in "ldapmodify Syntax" on page A-23.

> **See Also:** "Adding Configuration Set Entries by Using ldapadd" in *Oracle Internet Directory Administrator's Guide* for an explanation of using `ldapadd` to configure a server with an input file

`ldapadd` uses this syntax:

```
ldapadd [arguments] -f file_name
```

where *`file_name`* is the name of an LDIF file written with the specifications explained in the section "LDAP Data Interchange Format (LDIF) Syntax" on page A-1.

The following example adds the entry specified in the LDIF file `my_ldif_file.ldi`:

```
ldapadd -p 389 -h myhost -f my_ldif_file.ldi
```

Table A–7 Arguments for ldapadd

Optional Arguments	Description
`-b`	Specifies that you have included binary file names in the file, which are preceded by a forward slash character. The tool retrieves the actual values from the file referenced.
`-c`	Tells ldapadd to proceed in spite of errors. The errors will be reported. (If you do not use this option, `ldapadd` stops when it encounters an error.)
`-D` *`binddn`*	When authenticating to the directory, specifies doing so as the entry specified in *`binddn`*. This is the DN of the user seeking authentication. Use this with the `-w` *`password`* option.
`-E` *`character_set`*	Specifies native character set encoding. See the appendix about globalization support in *Oracle Internet Directory Administrator's Guide*.
`-f` *`file_name`*	Specifies the input name of the LDIF format import data file. For a detailed explanation of how to format an LDIF file, see "LDAP Data Interchange Format (LDIF) Syntax" on page A-1.
`-h` *`ldaphost`*	Connects to *`ldaphost`*, rather than to the default host, that is, your local computer. *`ldaphost`* can be a computer name or an IP address.
`-K`	Same as `-k`, but performs only the first step of the Kerberos bind
`-k`	Authenticates using Kerberos authentication instead of simple authentication. To enable this option, you must compile with `KERBEROS` defined. You must already have a valid ticket granting ticket.
`-M`	Instructs the tool to send the `ManageDSAIT` control to the server. The `ManageDSAIT` control instructs the server not to send referrals to clients. Instead a referral entry is returned as a regular entry.
`-n`	Shows what would occur without actually performing the operation
`-O` *`ref_hop_limit`*	Specifies the number of referral hops that a client should process. The default value is `5`.

Table A–7 (Cont.) Arguments for ldapadd

Optional Arguments	Description
`-p directory_port`	Connects to the directory on TCP port ***directory_port***. If you do not specify this option, the tool connects to the default port (389).
`-P wallet_password`	Specifies wallet password required for one-way or two-way SSL connections
`-U SSLAuth`	Specifies SSL authentication mode: ■ 1 for no authentication required ■ 2 for one way authentication required ■ 3 for two way authentication required
`-v`	Specifies verbose mode
`-V ldap_version`	Specifies the version of the LDAP protocol to use. The default value is 3, which causes the tool to use the LDAP v3 protocol. A value of 2 causes the tool to use the LDAP v2 protocol.
`-w password`	Provides the password required to connect
`-W wallet_location`	Specifies wallet location required for one-way or two-way SSL connections. For example, on UNIX, you could set this parameter as follows: `-W "file://home/my_dir/my_wallet"`. On Windows NT, you could set this parameter as follows: `-W "file:C:\my_dir\my_wallet"`.
`-X dsml_file`	Specifies the input name of the DSML format import data file.

ldapaddmt Syntax

`ldapaddmt` is like `ldapadd`: It enables you to add entries, their object classes, attributes, and values to the directory. It is unlike `ldapadd` in that it supports multiple threads for adding entries concurrently.

While it is processing LDIF entries, `ldapaddmt` logs errors in the `add.log` file in the current directory.

`ldapaddmt` uses this syntax:

```
ldapaddmt -T number_of_threads -h host -p port -f file_name
```

where ***file_name*** is the name of an LDIF file written with the specifications explained in the section "LDAP Data Interchange Format (LDIF) Syntax" on page A-1.

The following example uses five concurrent threads to process the entries in the file `myentries.ldif`.

```
ldapaddmt -T 5 -h node1 -p 3000 -f myentries.ldif
```

Note: Increasing the number of concurrent threads improves the rate at which LDIF entries are created, but consumes more system resources.

Table A–8 Arguments for ldapaddmt

Optional Arguments	Description
`-b`	Specifies that you have included binary file names in the data file, which are preceded by a forward slash character. The tool retrieves the actual values from the file referenced.
`-c`	Tells the tool to proceed in spite of errors. The errors will be reported. (If you do not use this option, the tool stops when it encounters an error.)
`-D` *`binddn`*	When authenticating to the directory, specifies doing so as the entry is specified in *`binddn`*—that is, the DN of the user seeking authentication. Use this with the `-w` *`password`* option.
`-E` *`character_set`*	Specifies native character set encoding. See the appendix about globalization support in *Oracle Internet Directory Administrator's Guide.*
`-h` *`ldap_host`*	Connects to *`ldaphost`*, rather than to the default host, that is, your local computer. *`ldaphost`* can be a computer name or an IP address.
`-K`	Same as `-k`, but performs only the first step of the kerberos bind
`-k`	Authenticates using Kerberos authentication instead of simple authentication. To enable this option, you must compile with `KERBEROS` defined. You must already have a valid ticket granting ticket.
`-M`	Instructs the tool to send the `ManageDSAIT` control to the server. The `ManageDSAIT` control instructs the server not to send referrals to clients. Instead a referral entry is returned as a regular entry.
`-n`	Shows what would occur without actually performing the operation.
`-O` *`ref_hop_limit`*	Specifies the number of referral hops that a client should process. The default value is `5.`
`-p` *`ldapport`*	Connects to the directory on TCP port *`ldapport`*. If you do not specify this option, the tool connects to the default port (`389`).
`-P` *`wallet_password`*	Specifies wallet password required for one-way or two-way SSL connections
`-T`	Sets the number of threads for concurrently processing entries
`-U` *`SSLAuth`*	Specifies SSL Authentication Mode: ■ `1` for no authentication required ■ `2` for one way authentication required ■ `3` for two way authentication required
`-v`	Specifies verbose mode
`-V` *`ldap_version`*	Specifies the version of the LDAP protocol to use. The default value is `3`, which causes the tool to use the LDAP v3 protocol. A value of `2` causes the tool to use the LDAP v2 protocol.
`-w` *`password`*	Provides the password required to connect
`-W` *`wallet_location`*	Specifies wallet location required for one-way or two-way SSL connections. For example, on UNIX, you could set this parameter as follows: `-W "file://home/my_dir/my_wallet"`. On Windows NT, you could set this parameter as follows: `-W "file:C:\my_dir\my_wallet"`.

Table A–8 (Cont.) Arguments for ldapaddmt

Optional Arguments	Description
`-X dsml_file`	Specifies the input name of the DSML format import data file.

ldapbind Syntax

The `ldapbind` command-line tool enables you to see whether you can authenticate a client to a server.

`ldapbind` uses this syntax:

```
ldapbind [arguments]
```

Table A–9 Arguments for ldapbind

Optional Arguments	Description
`-D binddn`	When authenticating to the directory, specifies doing so as the entry specified in `binddn`—that is, the DN of the user seeking authentication. Use this with the `-w password` option.
`-E character_set`	Specifies native character set encoding. See the appendix about globalization support in *Oracle Internet Directory Administrator's Guide*.
`-h ldaphost`	Connects to *`ldaphost`*, rather than to the default host, that is, your local computer. *`ldaphost`* can be a computer name or an IP address.
`-n`	Shows what would occur without actually performing the operation.
`-p ldapport`	Connects to the directory on TCP port *`ldapport`*. If you do not specify this option, the tool connects to the default port (`389`).
`-P wallet_password`	Specifies the wallet password required for one-way or two-way SSL connections.
`-U SSLAuth`	Specifies SSL authentication mode: `1` for no authentication required `2` for one way authentication required `3` for two way authentication required.
`-V ldap_version`	Specifies the version of the LDAP protocol to use. The default value is `3`, which causes the tool to use the LDAP v3 protocol. A value of `2` causes the tool to use the LDAP v2 protocol.
`-w password`	Provides the password required to connect.
`-W wallet_location`	Specifies wallet location required for one-way or two-way SSL connections. For example, on UNIX, you could set this parameter as follows: `-W file://home/my_dir/my_wallet`. On Windows NT, you could set this parameter as follows: `-W "file:C:\my_dir\my_wallet"`.
`-O sasl_properties`	Specifies SASL security properties. The security property supported is `-O "auth"`. This security property is for DIGEST-MD5 SASL mechanism. It enables authentication with no data integrity or data privacy.
`-Y sasl_mechanism`	Specifies a SASL mechanism. These mechanisms are supported: ■ `Y "DIGEST-MD5"` ■ `Y "EXTERNAL"`: The SASL authentication in this mechanism is done on top of two-way SSL authentication. In this case the identity of the user stored in the SSL wallet is used for SASL authentication.

Table A–9 (Cont.) Arguments for ldapbind

Optional Arguments	Description
`-R sasl_realm`	Specifies a SASL realm.

Table A–10 Optional Arguments

Optional Arguments	Description
`-O sasl_properties`	Specifies SASL security properties. The security property supported is `-O "auth"`. This security property is for DIGEST-MD5 SASL mechanism. It enables authentication with no data integrity or data privacy.
`-Y sasl_mechanism`	Specifies a SASL mechanism. These mechanisms are supported: ■ `Y "DIGEST-MD5"` ■ `Y "EXTERNAL"`: The SASL authentication in this mechanism is done on top of two-way SSL authentication. In this case the identity of the user stored in the SSL wallet is used for SASL authentication.
`-R sasl_realm`	Specifies a SASL Realm.

ldapcompare Syntax

The `ldapcompare` command-line tool enables you to match attribute values you specify in the command line with the attribute values in the directory entry.

`ldapcompare` uses this syntax:

```
ldapcompare [arguments]
```

The following example tells you whether `Person Nine`'s title is `associate`.

```
ldapcompare -p 389 -h myhost -b "cn=Person Nine,ou=EuroSInet Suite,o=IMC,c=US" -a
title -v associate
```

Table A–11 Arguments for ldapcompare

Optional Arguments	Description
`-a attribute_name`	Specifies the attribute on which to perform the compare. This argument is mandatory.
`-b basedn`	Specifies the distinguished name of the entry on which to perform the compare. This argument is mandatory.
`-v attribute_value`	Specifies the attribute value to compare. This argument is mandatory.
`-D binddn`	When authenticating to the directory, specifies doing so as the entry is specified in *`binddn`*—that is, the DN of the user seeking authentication. Use this with the `-w` *`password`* option.
`-d debug_level`	Sets the debugging level. See the chapter about logging, auditing, and monitoring in *Oracle Internet Directory Administrator's Guide*.
`-E character_set`	Specifies native character set encoding. See the appendix about globalization support in *Oracle Internet Directory Administrator's Guide*.
`-f file_name`	Specifies the input file name

Table A–11 (Cont.) Arguments for ldapcompare

Optional Arguments	Description
`-h ldaphost`	Connects to *ldaphost*, rather than to the default host, that is, your local computer. *ldaphost* can be a computer name or an IP address.
`-M`	Instructs the tool to send the `ManageDSAIT` control to the server. The `ManageDSAIT` control instructs the server not to send referrals to clients. Instead a referral entry is returned as a regular entry.
`-O ref_hop_limit`	Specifies the number of referral hops that a client should process. The default value is `5`.
`-p ldapport`	Connects to the directory on TCP port *ldapport*. If you do not specify this option, the tool connects to the default port (`389`).
`-P wallet_password`	Specifies wallet password required for one-way or two-way SSL connections
`-U SSLAuth`	Specifies SSL authentication mode: ■ `1` for no authentication required ■ `2` for one way authentication required ■ `3` for two way authentication required
`-V ldap_version`	Specifies the version of the LDAP protocol to use. The default value is `3`, which causes the tool to use the LDAP v3 protocol. A value of `2` causes the tool to use the LDAP v2 protocol.
`-w password`	Provides the password required to connect
`-W wallet_location`	Specifies wallet location required for one-way or two-way SSL connections. For example, on UNIX, you could set this parameter as follows: `-W "file://home/my_dir/my_wallet"`. On Windows NT, you could set this parameter as follows: `-W "file:C:\my_dir\my_wallet"`.

ldapdelete Syntax

The `ldapdelete` command-line tool enables you to remove entire entries from the directory that you specify in the command line.

`ldapdelete` uses this syntax:

```
ldapdelete [arguments] ["entry_DN" | -f input_file_name]
```

> **Note:** If you specify the entry DN, then do not use the `-f` option.

The following example uses port `389` on a host named `myhost`.

```
ldapdelete -p 389 -h myhost "ou=EuroSInet Suite, o=IMC, c=US"
```

Table A–12 Arguments for ldapdelete

Optional Argument	Description
`-D binddn`	When authenticating to the directory, uses a full DN for the *binddn* parameter—that is, the DN of the user seeking authentication; typically used with the `-w` *password* option.

Table A–12 (Cont.) Arguments for ldapdelete

Optional Argument	Description
`-d debug_level`	Sets the debugging level. See "Setting Debug Logging Levels by Using the OID Control Utility" in *Oracle Internet Directory Administrator's Guide*.
`-E character_set`	Specifies native character set encoding. See the appendix about globalization support in *Oracle Internet Directory Administrator's Guide*.
`-f input_file_name`	Specifies the input file name
`-h ldaphost`	Connects to *`ldaphost`*, rather than to the default host, that is, your local computer. *`ldaphost`* can be a computer name or an IP address.
`-k`	Authenticates using authentication instead of simple authentication. To enable this option, you must compile with Kerberos defined. You must already have a valid ticket granting ticket.
`-M`	Instructs the tool to send the `ManageDSAIT` control to the server. The `ManageDSAIT` control instructs the server not to send referrals to clients. Instead a referral entry is returned as a regular entry.
`-n`	Shows what would be done, but doesn't actually delete
`-O ref_hop_limit`	Specifies the number of referral hops that a client should process. The default value is `5`.
`-p ldapport`	Connects to the directory on TCP port *`ldapport`*. If you do not specify this option, the tool connects to the default port (`389`).
`-P wallet_password`	Specifies wallet password required for one-way or two-way SSL connections
`-U SSLAuth`	Specifies SSL authentication mode: ■ `1` for no authentication required ■ `2` for one way authentication required ■ `3` for two way authentication required
`-v`	Specifies verbose mode
`-V ldap_version`	Specifies the version of the LDAP protocol to use. The default value is `3`, which causes the tool to use the LDAP v3 protocol. A value of `2` causes the tool to use the LDAP v2 protocol.
`-w password`	Provides the password required to connect.
`-W wallet_location`	Specifies wallet location required for one-way or two-way SSL connections. For example, on UNIX, you could set this parameter as follows: `-W "file://home/my_dir/my_wallet"`. On Windows NT, you could set this parameter as follows: `-W "file:C:\my_dir\my_wallet"`.

ldapmoddn Syntax

The `ldapmoddn` command-line tool enables you to modify the DN or RDN of an entry.

`ldapmoddn` uses this syntax:

```
ldapmoddn [arguments]
```

The following example uses `ldapmoddn` to modify the RDN component of a DN from `cn=mary smith` to `cn=mary jones`. It uses port `389`, and a host named `myhost`.

```
ldapmoddn -p 389 -h myhost -b "cn=mary smith,dc=Americas,dc=imc,dc=com" -R
"cn=mary jones"
```

Table A–13 Arguments for ldapmoddn

Argument	Description
`-b` *basedn*	Specifies DN of the entry to be moved. This argument is mandatory.
`-D` *binddn*	When authenticating to the directory, do so as the entry is specified in *binddn*. This is the DN of the user seeking authentication. Use this with the `-w` *password* option.
`-E` *character_set*	Specifies native character set encoding. See the appendix about globalization support in *Oracle Internet Directory Administrator's Guide*.
`-f` *file_name*	Specifies the input file name
`-h` *ldaphost*	Connects to *ldaphost*, rather than to the default host, that is, your local computer. *ldaphost* can be a computer name or an IP address.
`-M`	Instructs the tool to send the `ManageDSAIT` control to the server. The `ManageDSAIT` control instructs the server not to send referrals to clients. Instead a referral entry is returned as a regular entry.
`-N` *newparent*	Specifies new parent of the RDN. Either this argument or the `-R` argument must be specified.
`-O` *ref_hop_limit*	Specifies the number of referral hops that a client should process. The default value is `5`.
`-p` *ldapport*	Connects to the directory on TCP port `ldapport`. If you do not specify this option, the tool connects to the default port (`389`).
`-P` *wallet_password*	Specifies wallet password required for one-way or two-way SSL connections
`-r`	Specifies that the old RDN is not retained as a value in the modified entry. If this argument is not included, the old RDN is retained as an attribute in the modified entry.
`-R` *newrdn*	Specifies new RDN. Either this argument or the -N argument must be specified.
`-U` *SSLAuth*	Specifies SSL authentication mode: `1` for no authentication required `2` for one way authentication required `3` for two way authentication required
`-V` *ldap_version*	Specifies the version of the LDAP protocol to use. The default value is `3`, which causes the tool to use the LDAP v3 protocol. A value of `2` causes the tool to use the LDAP v2 protocol.
`-w` *password*	Provides the password required to connect.
`-W` *wallet_location*	Specifies wallet location required for one-way or two-way SSL connections. For example, on UNIX, you could set this parameter as follows: `-W "file://home/my_dir/my_wallet"`. On Windows NT, you could set this parameter as follows: `-W "file:C:\my_dir\my_wallet"`.

ldapmodify Syntax

The `ldapmodify` tool enables you to act on attributes.

`ldapmodify` uses this syntax:

```
ldapmodify [arguments] -f file_name
```

where *`file_name`* is the name of an LDIF file written with the specifications explained the section "LDAP Data Interchange Format (LDIF) Syntax" on page A-1.

The list of arguments in the following table is not exhaustive. These arguments are all optional.

Table A–14 Arguments for ldapmodify

Argument	Description
`-a`	Denotes that entries are to be added, and that the input file is in LDIF format.
`-b`	Specifies that you have included binary file names in the data file, which are preceded by a forward slash character.
`-c`	Tells `ldapmodify` to proceed in spite of errors. The errors will be reported. (If you do not use this option, `ldapmodify` stops when it encounters an error.)
`-D` *`binddn`*	When authenticating to the directory, specifies doing so as the entry is specified in *`binddn`*—that is, the DN of the user seeking authentication. Use this with the `-w` *`password`* option.
`-E` *`character_set`*	Specifies native character set encoding. See the appendix about globalization support in *Oracle Internet Directory Administrator's Guide*.
`-h` *`ldaphost`*	Connects to `ldaphost`, rather than to the default host, that is, your local computer. `ldaphost` can be a computer name or an IP address.
`-M`	Instructs the tool to send the `ManageDSAIT` control to the server. The `ManageDSAIT` control instructs the server not to send referrals to clients. Instead a referral entry is returned as a regular entry.
`-n`	Shows what would occur without actually performing the operation.
`-o` *`log_file_name`*	Can be used with the `-c` option to write the erroneous LDIF entries in the logfile. You must specify the absolute path for the log file name.
`-O` *`ref_hop_limit`*	Specifies the number of referral hops that a client should process. The default value is `5`.
`-p` *`ldapport`*	Connects to the directory on TCP port `ldapport`. If you do not specify this option, the tool connects to the default port (`389`).
`-P` *`wallet_password`*	Specifies wallet password required for one-way or two-way SSL connections
`-U` *`SSLAuth`*	Specifies SSL authentication mode: ■ `1` for no authentication required ■ `2` for one way authentication required ■ `3` for two way authentication required
`-v`	Specifies verbose mode

Table A–14 (Cont.) Arguments for ldapmodify

Argument	Description
`-V ldap_version`	Specifies the version of the LDAP protocol to use. The default value is 3, which causes the tool to use the LDAP v3 protocol. A value of 2 causes the tool to use the LDAP v2 protocol.
`-w password`	Overrides the default, unauthenticated, null bind. To force authentication, use this option with the `-D` option.
`-W wallet_location`	Specifies wallet location required for one-way or two-way SSL connections. For example, on UNIX, you could set this parameter as follows: `-W "file://home/my_dir/my_wallet"`. On Windows NT, you could set this parameter as follows: `-W "file:C:\my_dir\my_wallet"`.

To run `modify`, `delete`, and `modifyrdn` operations using the `-f` flag, use LDIF for the input file format (see "LDAP Data Interchange Format (LDIF) Syntax" on page A-1) with the specifications noted in this section:

If you are making several modifications, then, between each modification you enter, add a line that contains a hyphen (-) only. For example:

```
dn: cn=Barbara Fritchy,ou=Sales,o=Oracle,c=US
changetype: modify
add: work-phone
work-phone: 510/506-7000
work-phone: 510/506-7001
-
delete: home-fax
```

Unnecessary space characters in the LDIF input file, such as a space at the end of an attribute value, will cause the LDAP operations to fail.

Line 1: Every change record has, as its first line, the literal `dn:` followed by the DN value for the entry, for example:

```
dn:cn=Barbara Fritchy,ou=Sales,o=Oracle,c=US
```

Line 2: Every change record has, as its second line, the literal `changetype:` followed by the type of change (`add, delete, modify, modrdn`), for example:

```
changetype: modify
```

or

```
changetype: modrdn
```

Format the remainder of each record according to the following requirements for each type of change:

- `changetype: add`

  Uses LDIF format (see "LDAP Data Interchange Format (LDIF) Syntax" on page A-1).

- `changetype: modify`

  The lines that follow this changetype consist of changes to attributes belonging to the entry that you identified previously in Line 1. You can specify three different types of attribute modifications—add, delete, and replace—which are explained next:

– **Add attribute values**. This option to changetype modify adds more values to an existing multi-valued attribute. If the attribute does not exist, it adds the new attribute with the specified values:

```
add: attribute name
attribute name: value1
attribute name: value2...
```

For example:

```
dn:cn=Barbara Fritchy,ou=Sales,o=Oracle,c=US
changetype: modify
add: work-phone
work-phone: 510/506-7000
work-phone: 510/506-7001
```

– **Delete values**. If you supply only the `delete` line, all the values for the specified attribute are deleted. Otherwise, if you specify an attribute line, you can delete specific values from the attribute:

```
delete: attribute name
[attribute name: value1]
dn: cn=Barbara Fritchy,ou=Sales,o=Oracle,c=US
changetype: modify
delete: home-fax
```

– Replace values. Use this option to replace all the values belonging to an attribute with the new specified set:

```
replace: attribute name
[attribute name: value1 ...]
```

If you do not provide any attributes with `replace`, the directory adds an empty set. It then interprets the empty set as a delete request, and complies by deleting the attribute from the entry. This is useful if you want to delete attributes that may or may not exist. For example:

```
dn: cn=Barbara Fritchy,ou=Sales,o=Oracle,c=US
changetype: modify
replace: work-phone
work-phone: 510/506-7002
```

- `changetype:delete`

  This change type deletes entries. It requires no further input, since you identified the entry in Line 1 and specified a changetype of delete in Line 2.

  For example:

```
dn: cn=Barbara Fritchy,ou=Sales,o=Oracle,c=US
changetype: delete
```

- `changetype:modrdn`

  The line following the change type provides the new relative distinguished name using this format:

```
newrdn: RDN
```

  For example:

```
dn: cn=Barbara Fritchy,ou=Sales,o=Oracle,c=US
changetype: modrdn
newrdn: cn=Barbara Fritchy-Blomberg
```

To specify an attribute as single-valued, include the keyword `SINGLE-VALUE` in the attribute definition entry in the LDIF file. Surround it with white space.

Example: Using ldapmodify to Add an Attribute

This example adds a new attribute called `myAttr`. The LDIF file for this operation is:

```
dn: cn=subschemasubentry
changetype: modify
add: attributetypes
attributetypes: (1.2.3.4.5.6.7 NAME 'myAttr' DESC 'New attribute definition'
EQUALITY caseIgnoreMatch SYNTAX
'1.3.6.1.4.1.1466.115.121.1.15' )
```

On the first line, enter the DN specifying where this new attribute is to be located. All attributes and object classes they are stored in `cn=subschemasubentry`.

The second and third lines show the proper format for adding a new attribute.

The last line is the attribute definition itself. The first part of this is the object identifier number: `1.2.3.4.5.6.7`. It must be unique among all other object classes and attributes. Next is the `NAME` of the attribute. In this case the attribute NAME is `myAttr`. It must be surrounded by single quotes. Next is a description of the attribute. Enter whatever description you want between single quotes. At the end of this attribute definition in this example are optional formatting rules to the attribute. In this case we are adding a matching rule of `EQUALITY caseIgnoreMatch` and a SYNTAX of `Directory String`. This example uses the object ID number of `1.3.6.1.4.1.1466.115.121.1.15` instead of the `SYNTAXES` name which is `Directory String`.

Put your attribute information in a file formatted like this example. Then run the following command to add the attribute to the schema of your Oracle directory server.

```
ldapmodify -h yourhostname -p 389 -D "orcladmin" -w "welcome" -v -f
/tmp/newattr.ldif
```

This `ldapmodify` command assumes that your Oracle directory server is running on port `389`, that your super user account name is `orcladmin`, that your super user password is `welcome` and that the name of your LDIF file is `newattr.ldif`. Substitute the host name of your computer where you see *`yourhostname`*.

If you are not in the directory where the LDIF file is located, then you must enter the full directory path to the file at the end of your command. This example assumes that your LDIF file is located in the `/tmp` directory.

ldapmodifymt Syntax

The `ldapmodifymt` command-line tool enables you to modify several entries concurrently.

`ldapmodifymt` uses this syntax:

```
ldapmodifymt -T number_of_threads [arguments] -f file_name
```

where *`file_name`* is the name of an LDIF file written with the specifications explained the section "LDAP Data Interchange Format (LDIF) Syntax" on page A-1.

> **See Also:** "ldapmodify Syntax" on page A-23 for additional formatting specifications used by ldapmodifymt

The following example uses five concurrent threads to modify the entries in the file `myentries.ldif`.

```
ldapmodifymt -T 5 -h node1 -p 3000 -f myentries.ldif
```

> **Note:** The `ldapmodifymt` tool logs error messages in the file `add.log`, which is located in the directory where you are running the command.

The arguments in the following table are all optional.

Table A–15 Arguments for ldapmodifymt

Argument	Description
`-a`	Denotes that entries are to be added, and that the input file is in LDIF format. (If you are running `ldapadd`, this flag is not required.)
`-b`	Specifies that you have included binary file names in the data file, which are preceded by a forward slash character.
`-c`	Tells ldapmodify to proceed in spite of errors. The errors will be reported. (If you do not use this option, ldapmodify stops when it encounters an error.)
`-D` *`binddn`*	When authenticating to the directory, specifies doing so as the entry is specified in *`binddn`*—that is, the DN of the user seeking authentication. Use this with the `-w` *`password`* option.
`-E` *`character_set`*	Specifies native character set encoding. See the appendix about globalization support in *Oracle Internet Directory Administrator's Guide*.
`-h` *`ldaphost`*	Connects to *`ldaphost`*, rather than to the default host, that is, your local computer. *`ldaphost`* can be a computer name or an IP address.
`-M`	Instructs the tool to send the `ManageDSAIT` control to the server. The `ManageDSAIT` control instructs the server not to send referrals to clients. Instead a referral entry is returned as a regular entry.
`-n`	Shows what would occur without actually performing the operation.
`-O` *`ref_hop_limit`*	Specifies the number of referral hops that a client should process. The default value is `5`.
`-p` *`ldapport`*	Connects to the directory on TCP port *`ldapport`*. If you do not specify this option, the tool connects to the default port (`389`).
`-P` *`wallet_password`*	Specifies wallet password required for one-way or two-way SSL connections
`-T`	Sets the number of threads for concurrently processing entries
`-U` *`SSLAuth`*	Specifies SSL authentication mode: ▪ `1` for no authentication required ▪ `2` for one way authentication required ▪ `3` for two way authentication required

Table A–15 (Cont.) Arguments for ldapmodifymt

Argument	Description
`-v`	Specifies verbose mode
`-V ldap_version`	Specifies the version of the LDAP protocol to use. The default value is `3`, which causes the tool to use the LDAP v3 protocol. A value of `2` causes the tool to use the LDAP v2 protocol.
`-w password`	Overrides the default, unauthenticated, null bind. To force authentication, use this option with the `-D` option.
`-W wallet_location`	Specifies wallet location required for one-way or two-way SSL connections. For example, on UNIX, you could set this parameter as follows: `-W "file://home/my_dir/my_wallet"`. On Windows NT, you could set this parameter as follows: `-W "file:C:\my_dir\my_wallet"`.

ldapsearch Syntax

The `ldapsearch` command-line tool enables you to search for and retrieve specific entries in the directory.

The `ldapsearch` tool uses this syntax:

```
ldapsearch [arguments] filter [attributes]
```

The *`filter`* format must be compliant with RFC-2254.

> **See Also:** RFC-2254 available at `http://www.ietf.org` for further information about the standard for the filter format

Separate attributes with a space. If you do not list any attributes, all attributes are retrieved.

Note:

- The `ldapsearch` tool does not generate LDIF output by default. To generate LDIF output from the `ldapsearch` command-line tool, use the `-L` flag.
- Various UNIX shells interpret some characters—for example, asterisks (*)—as special characters. Depending on the shell you are using, you may need to escape these characters.

Table A–16 Arguments for ldapsearch

Argument	Description
`-b basedn`	Specifies the base DN for the search. This argument is mandatory.
`-s scope`	This argument is mandatory. Specifies search scope: base, one, or sub Base: Retrieves a particular directory entry. Along with this search depth, you use the search criteria bar to select the attribute `objectClass` and the filter `Present`. One Level: Limits your search to all entries beginning one level down from the root of your search Subtree: Searches entries within the entire subtree, including the root of your search

Table A–16 (Cont.) Arguments for ldapsearch

Argument	Description
`-A`	Retrieves attribute names only (no values)
`-a deref`	Specifies alias dereferencing: never, always, search, or find
`-B`	Allows printing of non-ASCII values
`-D binddn`	When authenticating to the directory, specifies doing so as the entry specified in *`binddn`*—that is, the DN of the user seeking authentication. Use this with the `-w password` option.
`-d debug_level`	Sets debugging level to the level specified (see the chapter about logging, monitoring, and auditing in *Oracle Internet Directory Administrator's Guide*).
`-E character_set`	Specifies native character set encoding. See the appendix about globalization support in*Oracle Internet Directory Administrator's Guide*.
`-f file`	Performs sequence of searches listed in *`file`*.
`-F sep`	Prints`sep` instead of = between attribute names and values
`-h ldaphost`	Connects to *`ldaphost`*, rather than to the default host, that is, your local computer. *`ldaphost`* can be a computer name or an IP address.
`-L`	Prints entries in LDIF format (`-B` is implied).
`-l timelimit`	Specifies maximum time (in seconds) to wait for ldapsearch command to complete
`-M`	Instructs the tool to send the `ManageDSAIT` control to the server. The `ManageDSAIT` control instructs the server not to send referrals to clients. Instead a referral entry is returned as a regular entry.
`-n`	Shows what would be done without actually searching
`-O ref_hop_limit`	Specifies the number of referral hops that a client should process. The default value is `5`.
`-p ldapport`	Connects to the directory on TCP port *`ldapport`*. If you do not specify this option, the tool connects to the default port (`389`).
`-P wallet_password`	Specifies wallet password required for one-way or two-way SSL connections
`-S attr`	Sorts the results by attribute *`attr`*.
`-t`	Writes to files in `/tmp`.
`-u`	Includes user friendly entry names in the output
`-U SSLAuth`	Specifies the SSL authentication mode: ■ `1` for no authentication required ■ `2` for one way authentication required ■ `3` for two way authentication required
`-v`	Specifies verbose mode
`-w passwd`	Specifies bind passwd for simple authentication

Table A–16 (Cont.) Arguments for ldapsearch

Argument	Description
`-W wallet_location`	Specifies wallet location required for one-way or two-way SSL connections. For example, on UNIX, you could set this parameter as follows: `-W "file://home/my_dir/my_wallet"`.
	On Windows NT, you could set this parameter as follows: `-W "file:C:\my_dir\my_wallet"`.
`-z sizelimit`	Specifies maximum number of entries to retrieve
`-X`	Prints the entries in DSML v1 format.

Examples of ldapsearch Filters

Study the following examples to see how to build your own search commands.

Example 1: Base Object Search The following example performs a base-level search on the directory from the root.

```
ldapsearch -p 389 -h myhost -b "" -s base -v "objectclass=*"
```

- `-b` specifies base DN for the search, root in this case.
- `-s` specifies whether the search is a base search (`base`), one level search (`one`) or subtree search (`sub`).
- `objectclass=*` specifies the filter for search.

Example 2: One-Level Search The following example performs a one level search starting at `ou=HR, ou=Americas, o=IMC, c=US`.

```
ldapsearch -p 389 -h myhost -b "ou=HR, ou=Americas, o=IMC, c=US" -s one -v
"objectclass=*"
```

Example 3: Subtree Search The following example performs a subtree search and returns all entries having a DN starting with `cn=us`.

```
ldapsearch -p 389 -h myhost -b "c=US" -s sub -v "cn=Person*"
```

Example 4: Search Using Size Limit The following example actually retrieves only two entries, even if there are more than two matches.

```
ldapsearch -h myhost -p 389 -z 2 -b "ou=Benefits,ou=HR,ou=Americas,o=IMC,c=US" -s
one "objectclass=*"
```

Example 5: Search with Required Attributes The following example returns only the `DN` attribute values of the matching entries:

```
ldapsearch -p 389 -h myhost -b "c=US" -s sub -v "objectclass=*" dn
```

The following example retrieves only the distinguished name along with the surname (`sn`) and description (`description`) attribute values:

```
ldapsearch -p 389 -h myhost -b "c=US" -s sub -v "cn=Person*" dn sn description
```

Example 6: Search for Entries with Attribute Options The following example retrieves entries with common name (`cn`) attributes that have an option specifying a language code attribute option. This particular example retrieves entries in which the common names are in French and begin with the letter R.

```
ldapsearch -p 389 -h myhost -b "c=US" -s sub "cn;lang-fr=R*"
```

Suppose that, in the entry for John, no value is set for the `cn;lang-it` language code attribute option. In this case, the following example does not return John's entry:

```
ldapsearch -p 389 -h myhost -b "c=us" -s sub "cn;lang-it=Giovanni"
```

Example 7: Searching for All User Attributes and Specified Operational Attributes The following example retrieves all user attributes and the `createtimestamp` and `orclguid` operational attributes:

```
ldapsearch -p 389 -h myhost -b "ou=Benefits,ou=HR,ou=Americas,o=IMC,c=US" -s sub
"cn=Person*" * createtimestamp orclguid
```

The following example retrieves entries modified by Anne Smith:

```
ldapsearch -h sun1 -b "" "(&(objectclass=*)(modifiersname=cn=Anne
Smith))"
```

The following example retrieves entries modified between 01 April 2001 and 06 April 2001:

```
ldapsearch -h sun1 -b "" "(&(objectclass=*)(modifytimestamp >= 20000401000000)
(modifytimestamp <= 20000406235959))"
```

> **Note:** Because `modifiersname` and `modifytimestamp` are not indexed attributes, use `catalog.sh` to index these two attributes. Then, restart the Oracle directory server before issuing the two previous `ldapsearch` commands.

Other Examples: Each of the following examples searches on port `389` of host `sun1`, and searches the whole subtree starting from the DN `ou=hr,o=acme,c=us`.

The following example searches for all entries with any value for the `objectclass` attribute.

```
ldapsearch -p 389 -h sun1 -b "ou=hr, o=acme, c=us" -s subtree "objectclass=*"
```

The following example searches for all entries that have `orcl` at the beginning of the value for the `objectclass` attribute.

```
ldapsearch -p 389 -h sun1 -b "ou=hr, o=acme, c=us" -s subtree "objectclass=orcl*"
```

The following example searches for entries where the `objectclass` attribute begins with `orcl` and `cn` begins with foo.

```
ldapsearch -p 389 -h sun1 -b "ou=hr, o=acme, c=us" -s subtree
"(&(objectclass=orcl*)(cn=foo*))"
```

The following example searches for entries in which the common name (`cn`) is not `foo`.

```
ldapsearch -p 389 -h sun1 -b "ou=hr, o=acme, c=us" -s subtree "(!(cn=foo))"
```

The following example searches for entries in which `cn` begins with `foo` or `sn` begins with `bar`.

```
ldapsearch -p 389 -h sun1 -b "ou=hr, o=acme, c=us" -s subtree
"(|(cn=foo*)(sn=bar*))"
```

The following example searches for entries in which `employeenumber` is less than or equal to `10000`.

```
ldapsearch -p 389 -h sun1 -b "ou=hr, o=acme, c=us" -s subtree
"employeenumber<=10000"
```

Oracle Directory Integration and Provisioning Platform Command-Line Tools Syntax

This section contains these topics:

- The Directory Integration and Provisioning Assistant (dipassistant) Syntax
- The schemasync Tool Syntax
- The Oracle Directory Integration and Provisioning Server Registration Tool (odisrvreg)
- Syntax for Provisioning Subscription Tool (oidprovtool)

The Directory Integration and Provisioning Assistant (dipassistant) Syntax

The Directory Integration and Provisioning Assistant (`dipassistant`) is a command-line tool for administering the Oracle directory integration and provisioning server. The syntax for the Directory Integration and Provisioning Assistant is:

```
dipassistant [-gui | command] [-help]

command := createprofile [cp]
| createprofilelike [cpl]
| modifyprofile [mp]
| deleteprofile [dp]
| listprofiles[lp | lsprof]
| showprofile[sp]
| expressconfig[ec]
| bootstrap [bs]
| wpasswd [wp]
| chgpasswd [cpw]
| reassociate [rs]
```

For help on a particular command, enter:

```
dipassistant command -help
```

Table A–17 lists the tasks you can perform with the Directory Integration and Provisioning Assistant. It also points you to instructions for performing each task.

Table A–17 Summary of Functionality of the Directory Integration and Provisioning Assistant

Tasks	Commands	More Information
Use the Oracle Directory Integration and Provisioning Server Administration tool, which is the graphical version of the Directory Integration and Provisioning Assistant	`-gui`	The chapter about tools in *Oracle Identity Management Integration Guide*.

Create, modify, or delete a synchronization profile	`createprofile`
	`createprofilelike`
	`modifyprofile`
	`deleteprofile`
		"Creating, Modifying, and Deleting Synchronization Profiles" on page A-33

See all the profile names in Oracle Internet Directory	`listprofiles`	"Listing All Synchronization Profiles in Oracle Internet Directory" on page A-35

Table A–17 (Cont.) Summary of Functionality of the Directory Integration and Provisioning Assistant

Tasks	Commands	More Information
See the details of a specific profile	`showprofile`	"Viewing the Details of a Specific Synchronization Profile" on page A-36
Creates and configures import and export profiles for synchronization with Microsoft Active Directory	`expressconfig`	"Performing an Express Configuration of the Active Directory Connector Profiles" on page A-37
Make Oracle Internet Directory and the connected directory identical before beginning synchronization	`bootstrap`	"Bootstrapping a Directory by Using the Directory Integration and Provisioning Assistant" on page A-37
Set the wallet password that the Oracle directory integration and provisioning server later uses to connect to Oracle Internet Directory	`wpasswd`	"Setting the Wallet Password for the Oracle Directory Integration and Provisioning Server" on page A-41
Reset the password of the administrator of the Oracle Directory Integration Platform	`chgpasswd`	"Changing the Password of the Administrator of Oracle Directory Integration and Provisioning Platform" on page A-41
Move integration profiles from one identity management node to another	`reassociate`	"Moving an Integration Profile to a Different Identity Management Node" on page A-42

Creating, Modifying, and Deleting Synchronization Profiles

The syntax for creating, modifying, or deleting synchronization profiles by using the Directory Integration and Provisioning Assistant is:

```
dipassistant createprofile [-h hostName] [-p port] [-D bindDn] [-w password] -f
fileName -configset Configset Number

dipassistant createprofilelike [-h hostName] [-p port] [-D bindDn] [-w password]
-profile origProfName -newprofile newProfName

dipassistant modifyprofile [-h hostName] [-p port] [-D bindDn] [-w password]
{-f fileName | -profile profName [-updlcn] } [propName1=value]
[propName2=value]...

dipassistant deleteprofile -profile profName [-h hostName] [-p port] [-D bindDn]
[-w password] [-configset Configset Number]
```

Table A–18 describes the parameters for creating, modifying, and deleting synchronization profiles by using the Directory Integration and Provisioning Assistant.

Table A–18 Parameters for Creating, Modifying, and Deleting Synchronization Profiles by Using the Directory Integration and Provisioning Assistant

Parameter	Description
`-h | -host`	Host where Oracle Internet Directory is running. The default value is the name of the local host.
`-p | -port`	Port at which Oracle Internet Directory was started. The default is `389`.
`-D | -dn`	The bind DN to be used in identifying to the directory. The default value is the DN of the Oracle Directory Integration and Provisioning platform administrator.
`-w | -passwd`	The password of the bind DN to be used while binding to the directory.

Table A–18 (Cont.) Parameters for Creating, Modifying, and Deleting Synchronization Profiles by Using the Directory Integration and Provisioning Assistant

Parameter	Description
`-f | -file`	The configuration file containing the profile parameters. **See Also:** Table A–19 on page A-34 for a list of parameters and their description
`-configset`	An integer greater than `0` that represents the configuration set with which to associate the profile.
`-profile`	A text string representing the name of profile to be modified, deleted, or used as a template for creating a new profile.
`-newProfile | -name`	A text string representing the name of profile to be created in Oracle Internet Directory.
`-updlcn`	Updates the last applied changed number in the specified profile

The following example uses a configuration file named `import.profile` to create a new profile and associate the new profile with configuration set 1:

```
dipassistant createprofile -h myhost -p 3060 -D cn=dipadmin -w welcome1
-f import.profile -configset 1
```

The following example creates a new profile named `iPlImport` with values copied from a profile named `iPllmportTemplate`.

```
dipassistant createprofilelike -h myhost -p 3060 -D cn=dipadmin -w welcome1
-profile iPlImportTemplate -newProfile iPlImport
```

The following example uses a configuration file named `changes.profile` to modify a profile named `myprofile`.

```
dipassistant modifyprofile -profile myprofile -h myhost -p 3060 -D cn=dipadmin
-w welcome1 -f changes.profile
```

The following example deletes the `myprofile` profile.

```
dipassistant deleteprofile -profile myprofile -h myhost -p 3060 -D cn=dipadmin
-w welcome1 -configset 1
```

For the `createprofile`, `createprofilelike`, and `modifyprofile` commands, you specify a configuration file containing the properties listed in Table A–19. When modifying an already existing profile, no defaults are assumed. Only those attributes specified in the file are changed. When using Directory Integration and Provisioning Assistant, you reference a property name in the format `odip.profile.`*`property_name`*. However, in Oracle Internet Directory, the property name is stored in the format `orclodip`*`property_name`*. Both property name formats are listed in Table A–19.

Table A–19 Properties Expected by createprofile and modifyprofile Commands

Property	Description	Default
`odip.profile.agentexecommand / orclodipagentexecommand`	In the case of a non-LDAP interface, the command to produce the information in LDIF format	-
`odip.profile.condiraccount / orclodipcondiraccessaccount`	DN or user name used to connect to the third party directory.	-
`odip.profile.condirfilter / orclodipcondirmatchingfilter`	Filter that needs to be applied to the changes read from the connected directory before importing to Oracle Internet Directory	-

Table A–19 (Cont.) Properties Expected by createprofile and modifyprofile Commands

Property	Description	Default
odip.profile.condirpassword / orclodipcondiraccesspassword	Password used for identification to the third-party directory.	-
odip.profile.condirurl / orclodipcondirurl	Location of third-party directory [hostname:port]	-
odip.profile.configfile	Name of the file that contains the additional profile-specific information to be used for execution	-
odip.profile.configinfo / orclodipadditionalconfiginfo	Contains additional profile-specific information to be used for execution	-
odip.profile.debuglevel / orclodipprofiledebuglevel	Specifies the profile debug level	-
odip.profile.interface / orclodipinterfacetype	Indicator as to whether the LDAP or LDIF or DB or TAGGED format is to be used for data exchange	LDAP
odip.profile.lastchgnum / orclodipcondirlastappliedchangenumber	Last applied change number. In the case of an export profile this number refers to Oracle Internet Directory's last applied change number However, n the case of the import profile, this number refers to the last applied change number in the connected directory	-
odip.profile.mapfile / orclodipattributemappingrules	Name of the file that contains the mapping rules	-
odip.profile.name / orclodipagentname	Name of the profile	-
odip.profile.oidfilter / orclodipoidmatchingfilter	Filter that needs to be applied to the changes that are read from the Oracle Internet Directory before exporting to the connected directory	-
odip.profile.password / orclODIPAgentPassword	Password for accessing this profile	-
odip.profile.retry / orclodipsyncretrycount	Maximum number of times the Oracle directory integration and provisioning server should attempt to execute an entry	4
odip.profile.schedinterval/ orclodipschedulinginterval	Interval between successive executions of this profile by the integration server. If the previous execution has not completed then the next execution will not resume until it completes.	One minute
odip.profile.status / orclodipagentcontrol	Either DISABLE or ENABLE	DISABLE
odip.profile.syncmode / orclodipasynchronizationmode	Direction of synchronization. When the changes are propagated from the third party to Oracle Internet Directory, the synchronization mode is IMPORT. When the changes are propagated to the third party directory, the synchronization mode is EXPORT.	IMPORT

Listing All Synchronization Profiles in Oracle Internet Directory

The `listprofiles` command prints a list of all the synchronization profiles in Oracle Internet Directory. The syntax for this command is:

```
dipassistant listprofiles [-h hostName] [-p port] [-D bindDn] [-w password]
[-configset Configset Number]
```

Table A–20 on page A-36 describes the parameters of the `listprofiles` command.

Table A–20 Parameters of the listprofiles Command

Parameter	Description
`-h | -host`	Host where Oracle Internet Directory is running. The default value is the name of the local host.
`-p | -port`	Port at which Oracle Internet Directory was started. The default is `389`.
`-D | -dn`	The bind DN to be used in identifying to the directory. The default value is the DN of the Oracle Directory Integration and Provisioning platform administrator.
`-w | -passwd`	The password of the bind DN to be used while binding to the directory.
`-configset`	An integer greater than 0 that represents the configuration set with which to associate the profile.

The following example prints a list of all the synchronization profiles in Oracle Internet Directory:

```
dipassistant listprofiles -h myhost -p 3060 -D cn=dipadmin -w welcome1
```

By default, the preceding command prints the following list of sample profiles created during installation. However, your deployment of Oracle Internet Directory may contain additional synchronization profiles.

```
IplanetExport
IplanetImport
ActiveImport
ActiveExport
LdifExport
LdifImport
TaggedExport
TaggedImport
OracleHRAgent
ActiveChgImp
```

Viewing the Details of a Specific Synchronization Profile

The `showprofile` command prints the details of a specific synchronization profile. The syntax for this command is:

```
dipassistant showprofile -profile profName [-h hostName] [-p port] [-D bindDn]
[-w password]
```

Table A–21 describes the parameters of the `showprofile` command.

Table A–21 Parameters of the showprofile Command

Parameter	Description
`-h | -host`	Host where Oracle Internet Directory is running. The default value is the name of the local host.
`-p | -port`	Port at which Oracle Internet Directory was started. The default is `389`.
`-D | -dn`	The bind DN to be used in identifying to the directory. The default value is the DN of the Oracle Directory Integration and Provisioning platform administrator.
`-w | -passwd`	The password of the bind DN to be used while binding to the directory.
`-profile`	A text string representing the name of profile to show.

For example, the following `showprofile` command prints the details for the `ActiveImport` sample profile that is created during installation:

```
dipassistant showprofile -h myhost -p 3060 -D cn=dipadmin -w welcome1
-profile ActiveImport
```

The preceding command prints the following details of the `ActiveImport` sample profile:

```
odip.profile.version = 2.0
odip.profile.lastchgnum = 0
odip.profile.interface = LDAP
odip.profile.oidfilter = orclObjectGUID
odip.profile.schedinterval = 60
odip.profile.name = ActiveImport
odip.profile.syncmode = IMPORT
odip.profile.condirfilter =
"searchfilter=(|(objectclass=group)(objectclass=organizationalunit)
(&(objectclass=user)(!(objectclass=computer))))"
odip.profile.retry = 5
odip.profile.debuglevel = 0
odip.profile.status = DISABLE
```

Performing an Express Configuration of the Active Directory Connector Profiles

The `expressconfig` command performs an express configuration of the Active Directory connector. When you run this command, it performs all required configurations outlined in Table A–17, " Summary of Functionality of the Directory Integration and Provisioning Assistant" on page A-32. This command also creates two profiles, an import profile and an export profile. The syntax for performing an express configuration is as follows:

```
dipassistant expressconfig [-h hostName] [-p port] [-3rdpartyds 3rd party ds]
[-configset Configset Number]
```

Table A–22 describes the parameters of the `expressconfig` command.

Table A–22 Parameters of the expressconfig Command

Parameter	Description
`-h | -host| -oidhost`	Host where Oracle Internet Directory is running. The default value is the name of the local host.
`-p |-port | -oidport`	Port at which Oracle Internet Directory was started. The default is `389`.
`-3rdpartyds`	The third-party directory service to configure.
`-configset`	An integer greater than 0 that represents the configuration set with which to associate the profile.

Bootstrapping a Directory by Using the Directory Integration and Provisioning Assistant

The `bootstrap` command performs the initial migration of data between a connected directory and Oracle Internet Directory. The syntax for this command is as follows:

```
dipassistant bootstrap { -profile profName [-h hostName] [-p port] [-D bindDn] [-w
password] [-log logFile] [-logseverity severity] [-trace traceFile] [-tracelevel
level] [-loadparallelism #nThrs] [-loadretry retryCnt] | -f filename }
```

Table A–23 on page A-38 describes the parameters of the `bootstrap` command.

Table A–23 Parameters of the bootstrap Command

Parameter	Description
`-f | cfg`	A configuration file containing all the parameters required for performing the bootstrapping. **See Also:** Table A–24 on page A-39 for a list of parameters and their description.
`-h | -host`	Host where Oracle Internet Directory is running. The default value is the name of the local host.
`-p | -port`	Port at which Oracle Internet Directory was started. The default is `389`.
`-D | -dn`	The bind DN to be used in identifying to the directory. The default value is the DN of the Oracle Directory Integration and Provisioning platform administrator.
`-w | -passwd`	The password of the bind DN to be used while binding to the directory.
`-profile`	A text string representing the name of profile to use when performing the bootstrapping.
`-log`	Log file. If this parameter is not specified, then, by default, the log information is written to *`ORACLE_HOME`*`/ldap/odi/bootstrap.log`
`-logseverity`	Log severity 1 - 15. `1` – INFO, `2` – WARNING, `3` – DEBUG, `4` – ERROR. Or any combination of these. If not specified, then INFO and ERROR messages alone will be logged.
`-trace`	Trace file for debugging purposes.
`-tracelevel`	Trace level.
`-loadparallelism`	Indicator that loading to Oracle Internet Directory is to take place in parallel by using multiple threads. For example, `-loadparallelism 5` means that five threads are to be created, each of which tries to load the entries in parallel to Oracle Internet Directory.
`-loadretry`	When loading to the destination fails, the number of times to retry before marking the entry bad.

When you use the `bootstrap` command, you can use either the `-profile` parameter to specify a synchronization profile or the `-f` parameter to a configuration file. The following example uses a synchronization profile named `iPlanetProfile` to perform bootstrapping:

```
dipassistant bootstrap –profile iPlanetProfile -h myhost –port 3060 -D cn=dipadmin
-w welcome1
```

The following example uses a configuration file named `bootstrap.cfg` to perform bootstrapping:

```
dipassistant bootstrap –f bootstrap.cfg
```

When you use the `-f` parameter with the `bootstrap` command, you must specify a configuration file containing the properties listed in Table A–24 on page A-39.

Properties Expected by the Bootstrapping Command

Table A–24 Bootstrapping Configuration File Properties

Property	Description	Mandatory	Default
`odip.bootstrap.srctype`	Indicator of whether source of the bootstrapping is LDAP or LDIF. Valid values are either `LDAP` or `LDIF`.	Yes	-
`odip.bootstrap.desttype`	Indicator of whether destination of the bootstrapping is LDAP or LDIF. Valid values are either `LDAP` or `LDIF`.	Yes	-
`odip.bootstrap.srcurl`	In the case of LDAP source type, location of the source directory. In the case of LDIF, the location of the LDIF file. **Note:** For LDAP, the expected format is `host[:port]`. For LDIF, the expected format is the absolute path of the file.	Yes	-
`odip.bootstrap.desturl`	In the case of LDAP, location of the destination directory. In the case of LDIF, the location of the LDIF file. **Note:** For LDAP, the expected format is `host[:port]`. For LDIF, the expected format is the absolute path of the file.	Yes	-
`odip.bootstrap.srcsslmode`	Indicator of whether SSL-based authentication must be used to connect to the source of the bootstrapping. A value of `TRUE` indicates that SSL-based authentication must be used.	No	`FALSE`
`odip.bootstrap.destsslmode`	Indicator of whether SSL-based authentication must be used to connect to the destination of the bootstrapping. `TRUE` indicates that SSL-based authentication must be used. **Note:** In the case of LDIF, this parameter is meaningless.	No	`FALSE`

Table A–24 (Cont.) Bootstrapping Configuration File Properties

Property	Description	Mandatory	Default
`odip.bootstrap.srcdn`	Supplement to the source URL. In the case of LDIF binding, this parameter is meaningless. However in the case of LDAP, this parameter specifies the Bind DN.	Only in the case of LDAP	-
`odip.bootstrap.destdn`	Supplement to the destination URL. In the case of LDIF binding, this parameter is meaningless. However in the case of LDAP, this parameter specifies the Bind DN.	Only in the case of LDAP	-
`odip.bootstrap.srcpasswd`	Bind password to the source. In the case of LDAP binding, this is used as security. Oracle Corporation recommends that you not specify the password in this file.	No	-
`odip.bootstrap.destpasswd`	Bind password. In the case of LDAP binding, this is used as security credential. Oracle Corporation recommends that you not specify the password in this file.	No	-
`odip.bootstrap.mapfile`	Location of the map file that contains the attribute and domain mappings.	No	-
`odip.bootstrap.logfile`	Location of the log file. If this file already exists then it will be appended. The default log file is `bootstrap.log` created under $*`ORACLE_HOME`*`/ldap/odi/log directory.`	No	The file `bootstrap.log` created under the directory *`ORACLE_HOME`*`/ldap/odi/`
`odip.bootstrap.logseverity`	Type of log messages that needs to be logged. INFO – `1` WARNING - `2` DEBUG – `4` ERROR - `8` **Note:** A combination of these types can also be given. For example, if you are interested only in WARNING and ERROR message, then specify a value of `8+2`—that is, `10`. Similarly, for all types of message, use `1 + 2 + 4 + 8 = 15`	No	`1 + 8 = 9`

Table A–24 (Cont.) Bootstrapping Configuration File Properties

Property	Description	Mandatory	Default
odip.bootstrap.loadparallelism	Numeric value indicating the number of writer threads used to load the processed data to the destination	No	1-
odip.bootstrap.loadretry	In the event of a failure to load an entry, indicator of how many times to retry	No	5
odip.bootstrap.trcfile	Location of the trace file. If this file already exists, then it is overwritten.	No	The default location is *ORACLE_HOME*/ldap/odi/log/bootstrap.trc
odip.bootstrap.trclevel	The tracing level	No	3
odip.bootstrap.srcencode	The encoding used by the LDIF file if the file: Is generated by using a utility of a third-party directory Contains NLS data Is processed on a different platform By default, the Directory Integration and Provisioning Assistant assumes that the file is processed on the system on which it was generated.	Yes	-

Setting the Wallet Password for the Oracle Directory Integration and Provisioning Server

The `wp` command enables you to set the wallet password that the Oracle directory integration and provisioning server later uses to connect to Oracle Internet Directory. To use this command, enter:

```
dipassistant wp
```

The Directory Integration and Provisioning Assistant prompts you to enter, and then confirm, the password.

Changing the Password of the Administrator of Oracle Directory Integration and Provisioning Platform

This `chgpasswd` command resets the password of `dipadmin` account. The default password for the `dipadmin` account is same as `ias_admin` password chosen during installation. To reset the password, you must provide the security credentials of the `orcladmin` account. The syntax for resetting the password is as follows:

```
dipassistant chgpasswd [-h hostName] [-p port] [-D bindDn] [-w password]
```

Table A–25 on page A-42 describes the parameters of the `chgpasswd` command.

Table A–25 Parameters of the chgpasswd Command

Parameter	Description
`-h | -host`	Host where Oracle Internet Directory is running. The default value is the name of the local host.
`-p | -port`	Port at which Oracle Internet Directory was started. The default is `389`.
`-D | -dn`	The bind DN to be used in identifying to the directory. The default value is the DN of the Oracle Directory Integration and Provisioning platform administrator.
`-w | -passwd`	The password of the bind DN to be used while binding to the directory.

The following is an example of the `chgpasswd` command:

```
dipassistant chgpasswd -h myhost -p 3060 -D cn=dipadmin -w welcome1
```

The Directory Integration and Provisioning Assistant then prompts for the new password as follows:

```
New Password:
Confirm Password:
```

Moving an Integration Profile to a Different Identity Management Node

You can use the `reassociate` command of the Directory Integration and Provisioning Assistant to move directory integration profiles to another node and to reassociate them with it. For example, if the middle-tier components are associated with a particular Oracle Identity Management infrastructure, then all the integration profiles existing in that infrastructure node can be moved to a new infrastructure node and reassociated with it.

Table A–26 describes the reassociation rules.

Table A–26 Scenarios for Reassociating Directory Integration Profiles

Scenario	Actions Taken
Integration profile does not exist on the second Oracle Internet Directory node	The integration profile is copied to the second Oracle Internet Directory node and is disabled after copying. It must be enabled by the application. The `lastchangenumber` attribute in the integration profile is modified to the current last change number on the second Oracle Internet Directory node.
Integration profile exists on the second Oracle Internet Directorynode	Both integration profiles are reconciled in the following manner: ▪ Any new attribute in the profile on node 1 is added to the profile on node 2 ▪ For existing same attributes, the values in profile on node 1 override the attributes in the profile on node 2 ▪ The profile is disabled after copying. It needs to be enabled by the application. ▪ The `lastchangenumber` attribute in the integration profile is modified to the current last change number on the second Oracle Internet Directory node

The syntax for the `reassociate` command is as follows:

```
dipassistant reassociate [-src_ldap_host hostName] [-src_ldap_port port] [-src_
ldap_dn bindDn] [-src_ldap_passwd password] -dst_ldap_host hostName [-dst_ldap_
port port [-dst_ldap_dn bindDn] [-dst_ldap_passwd password] [-log logfile]
```

Table A–27 describes the parameters of the `reassociate` command.

Table A–27 Parameters of the reassociate Command

Parameter	Description
`-src_ldap_host host_name`	Host where Oracle Internet Directory-1 runs
`-src_ldap_port port_number`	Port where Oracle Internet Directory-1 runs
`-src_ldap_dn bind_DN`	Bind DN for connecting to Oracle Internet Directory-1
`-src_ldap_passwd password`	Bind DN password for connecting to Oracle Internet Directory-1
`-dst_ldap_host host_name`	Host where Oracle Internet Directory-2 runs
`-dst_ldap_portport_number`	Port where Oracle Internet Directory-2 runs
`-dst_ldap_dn bind_DN`	Bind DN for connecting to Oracle Internet Directory-2
`-dst_ldap_passwd password`	Bind DN password for connecting to Oracle Internet Directory-2
`-log log_file`	Log file

The `reassociate` command defaults are as follows:

```
src_ldap_host - localhost, src_ldap_port & dst_ldap_port - 389
src_ldap_dn & dst_ldap_dn - cn=orcladmin account
```

The following is an example of the `reassociate` command:

```
dipassistant reassociate -src_ldap_host oid1.mycorp.com \
-dst_ldap_host oid2.mycorp.com -src_ldap_passwd srcpassword \
-dst_ldap_passwd dstpassword
```

Note if the location of the log file is not specified then by default it will be created as *`ORACLE_HOME`*`/ldap/odi/log/reassociate.log`.

Limitations of the Directory Integration and Provisioning Assistant in Oracle Internet Directory 10*g* Release 2 (10.1.2)

In this release, the Directory Integration and Provisioning Assistant does not support the following:

- SSL-based authentications to Oracle Internet Directory
- Schema synchronization
- Automatic profile creation at the end of the bootstrapping process when used with the `-cfg` option
- Mapping file validation
- Creation of a failed entries file

The following elements of the Directory Integration and Provisioning Assistant are untested:

- Bootstrapping of the connected directory over the SSL connection
- The use of the modifyprofile command while synchronization is happening for that profile

The bootstrapping command of the Directory Integration and Provisioning Assistant has the limitations described in Table A–28 on page A-44.

Table A–28 Limitations of Bootstrapping in the Directory Integration and Provisioning Assistant

Type of Bootstrapping	Limitation
LDIF-to-LDIF	None
LDAP-to-LDIF	For a large number of entries, bootstrapping can fail with an error of size limit exceeded. To resolve this, the connected directory server from which you are bootstrapping should: ■ Support paged results control (OID 1.2.840.113556.1.4.319). Currently, Microsoft Active Directory is the only LDAP directory that supports this control. ■ Have an adequate value for the server side search size limit parameter. ■ Use a proprietary tool on the connected directory server to dump all entries to an LDIF file, and then bootstrap by using either the LDIF-to-LDIF or the LDIF-to-LDAP approach.
LDIF -to-LDAP	None
LDAP-to-LDAP	Same as LDAP-to-LDIF

For initial bootstrapping, you should perform the following steps:

1. Generate a dump of the entries in the connected directory to an LDIF file using a proprietary tool on the connected directory server.
2. Configure the properties file so that entries are created in Oracle Internet Directory using the LDIF-to-LDAP approach.

The schemasync Tool Syntax

The `schemasync` tool enables you to synchronize schema elements—namely attributes and object classes—between an Oracle directory server and third-party LDAP directories.

The usage for `schemasync` is as follows:

```
ORACLE_HOME/bin/schemasync -srchost source_LDAP_directory -srcport
source_LDAP_port_number -srcdn privileged_DN_in_source_directory_to_access_schema
-srcpwd password -dsthost destination_directory -dstport destination_port
-dstdn privileged_dn_in_destination_directory -dstpwdpassword [-ldap]
```

> **Note:** the `-ldap` parameter is optional. If it is specified, then the schema changes are applied directly from the source LDAP directory to the destination LDAP directory. If it is not specified, then the schema changes are placed in the following LDIF files:
>
> - *ORACLE_HOME*`/ldap/odi/data/attributetypes.ldif`
> This file has the new attribute definitions.
> - *ORACLE_HOME*`/ldap/odi/data/objectclasses.ldif`
> This file has the new object class definitions.
>
> if you do not specify `-ldap`, then you must use `ldapmodify` to upload the definitions from these two files, first attribute types and then object classes.

The errors that occur during schema synchronization are logged in the following log files:

- *ORACLE_HOME*/ldap/odi/log/attributetypes.log
- *ORACLE_HOME*/ldap/odi/log/objectclasses.log

The Oracle Directory Integration and Provisioning Server Registration Tool (odisrvreg)

To register an Oracle directory integration and provisioning server with the directory, this tool creates an entry in the directory and sets the password for the directory integration and provisioning server. If the registration entry already exists, then you can use the tool to reset the existing password. The `odisrvreg` tool also creates a local file called `odisrvwallet_`*`hostname`*, at *ORACLE_HOME*`/ldap/odi/conf`. This file acts as a private wallet for the directory integration and provisioning server, which uses it on startup to bind to the directory.

Table A–29 describes the parameters that you use with the Oracle Directory Integration and Provisioning Server Registration Tool. You can also run `odisrvreg` in SSL mode to make communication between the tool and the directory fully secure, using the `-U`, `-W`, and `-P` parameters that are also described in Table A–29.

To register the directory integration and provisioning server, enter this command:

```
odisrvreg -h host_name -p port -D binddn -w bindpasswd –I passwd [-U ssl_mode -W
wallet –P wallet_password]
```

Table A–29 Descriptions of ODISRVREG Arguments

Argument	Description
`-h` *host_name*	Oracle directory server host name
`-p` *port_number*	Port number on which the directory server is running
`-D` *binddn*	Bind DN. The bind DN must have authorization to create the registration entry for the directory integration and provisioning server
`-l host`	In a cold failover cluster configuration, the virtual hostname
`-w bindpasswd`	Bind password
`-U` *ssl_mode*	For no authorization, specify `0`. For one-way authorization, specify `1`.
`-W` *wallet_location*	Location of the Oracle Wallet containing the SSL certificate
`-P` *wallet_password*	Wallet password to open the Oracle wallet

Syntax for Provisioning Subscription Tool (oidprovtool)

Use the Provisioning Subscription Tool (`oidprovtool`) to administer directory entries for provisioning profiles. You can perform these tasks:

- Create a profile (`create`)
- Disable a profile (`disable`)
- Reenable a profile (`enable`)
- Modify a profile (`modify`)
- Delete a profile (`delete`)
- Get the current status of a profile (`status`)
- Clear all of the errors in a profile (`reset`)

`oidprovtool` hides the location of profile entries from callers of the tool. It also hides schema details. From the caller's perspective, the combination of an application identity and an identity management realm uniquely identify a provisioning profile. There can be only one provisioning profile per application per identity management realm.

Note: To run `oidprovtool` and other shell scripts on Windows platforms, use one of these UNIX emulation utilities:

- Cygwin 1.0:

  `http://sources.redhat.com`

- MKS Toolkit 5.1 or 6.0:

  `http://www.datafocus.com/products`

The name of the executable is `oidProvTool`. It is found at *`ORACLE_HOME`*`/bin`. To invoke the tool, issue this command:

```
oidprovtool param1=param1_value  param2=param2_value param3=param3_value ...
```

`oidprovtool` accepts the following parameters:

Table A–30 Provisioning Subscription Tool Parameters

Name	Description	Operations	Mandatory/Optional
`operation`	The subscription operation performed. Only one operation can be performed for each invocation of the tool.	All	M
`ldap_host`	Host name of the directory server on which the operation is performed. If not specified, the default value of `localhost` is assumed.	All	O
`profile_status`	The status of the profile. This value can be either `enabled` or `disabled`. The default is `enabled`.	Create	O
`profile_mode`	The values possible are `inbound`, `outbound`, or both. The default is `outbound`.	Create	O
`profile_debug`	The debug level at which the DIP server executes the profile.	All	O
`sslmode`	A value of `0` indicates non-SSL. A value of `1` indicates SSL.	All	O
`ldap_port`	The TCP/IP port on which the directory server is listening for requests. Defaults to `389` if no value is specified.	All	O

Table A–30 (Cont.) Provisioning Subscription Tool Parameters

Name	Description	Operations	Mandatory/Optional
`ldap_user_dn`	The distinguished name of the user on whose behalf the operation is performed. The default is `cn=orcladmin`. Not all users may have the permissions necessary to perform provisioning subscription operations. See the administration guide to learn how to grant or deny permissions.	All	O
`ldap_user_password`	The password of the user on whose behalf the operation is performed. The default is `welcome`.	All	O
`application_dn`	The distinguished name of the application for which the operation is performed. Used together, `application_dn` and `organization_dn` help the subscription tool identify a provisioning profile.	All	M
`organization_dn`	The distinguished name of the organization for which the operation is performed. The default is the default identity management realm. Used together, `application_dn` and `organization_dn` help `oidprovtool` identify a provisioning profile.	All	O
`interface_name`	Database schema name for the PL/SQL package. The format of this value should be `[`*`Schema`*`].[`*`PACKAGE_NAME`*`]`	Create or modify	M
`interface_type`	The type of the interface to which events have to be propagated. The default is `PLSQL` if no value is specified.	Create or modify	O
`interface_connect_info`	The database connect string. The format is *`host`*`:`*`port`*`:`*`database_sid`*`:`*`user_id`*`:`*`password`*.	Create or modify	M
`interface_version`	The version of the interface protocol. Valid Values are 1.0, 1.1, or 2.0. Version 1.0 and 1.1 are old interfaces. Version 2.0 is the default.	Create	O
`interface_additional_info`	Additional information about the interface. This parameter is not used currently.	Create or modify	O
`schedule`	The length of time, in seconds, that must elapse before DIP processes the profile. Defaults to `3600` if no value is specified.	Create or modify	O

Table A–30 (Cont.) Provisioning Subscription Tool Parameters

Name	Description	Operations	Mandatory/Optional
`max_retries`	The number of times the provisioning service should try to deliver an event if delivery has failed. Defaults to `5` if no value is specified.	Create or modify	O
`max_events_per_schedule`	The maximum number of events that should be propagated in one schedule. The default is `100`. Useful for controlling load situations.	Create or modify	O
`profile_group`	The group of the profile. The Default is `0`. Use this parameter to address scalability issues that arise when different DIP server instances execute different groups.	Create or modify	O
`lastchangenumber`	The change number at which events are propagated to the application. Used only in `outbound` mode. The default for the `create` operation is the current number.	Create or modify	O

Table A–30 (Cont.) Provisioning Subscription Tool Parameters

Name	Description	Operations	Mandatory/Optional
`event_subscription`	Events that DIP should notify the application about. The string must have this format: `[USER | GROUP]:[`*`domain_of_interest`*`]:[DELETE | MODIFY(`*`list of attributes separated by commas`*`)]` You can specify multiple values by including more than one `event_subscription` parameter in a run of `oidprovtool`.	create only	M (for outbound mode only)

`event_mapping_rules`	For multivalued inbound events only. This parameter maps the type of object received from an application and a qualifying filter condition to determine the domain of interest for events indicated in `event_subscription`. The parameter takes this format:
	`OBJECT_TYPE:`*`filter_condition`*`:`*`domain_of_interest`*
	Multiple rules are allowed. You might, for example, enter these two rules:
	`EMP::cn=users,dc=acme,dc=com`
	`EMP:1=AMERICA:1=AMER,cn=users,dc=acme,dc=com`
	In the first case, if the object type received is `EMP`, the event is meant for the domain `cn=users,dc=acme,dc=com`. In the second case, the object received is again `EMP`, but the rule contains the attribute `l` (locality). The value of this attribute is `AMERICA`. Accordingly, the events specified are meant for the domain `l=AMER,cn=users,dc=acme,dc=com`.
		Create or modify	M (for inbound mode only)

Table A–30 (Cont.) Provisioning Subscription Tool Parameters

Name	Description	Operations	Mandatory/Optional
event_permitted_operations	For multivalued inbound events only. This parameter is used to define the types of events an application is privileged to send to the Provisioning Integration Service. The parameter takes this format: EVENT_OBJECT:affected_domain:operation(attributes) Here are two examples: IDENTITY:cn=users,dc=acme,dc=com:ADD(*) IDENTITY:cn=users,dc=acme,dc=com:MODIFY(cn,sn,mail,telephonenumber) In the first example, the IDENTITY_ADD event is allowed for the domain specified as well as all of its attributes. In the second example, IDENTITY_ADD is allowed for the same domain, but only for certain attributes. Other attributes are ignored.	Create or modify	M (for inbound mode only)

B

DSML Syntax

This appendix contains the following sections:

Capabilities of DSML

Directory services form a core part of distributed computing. XML is becoming the standard markup language for Internet applications. As directory services are brought to the Internet, there is a pressing and urgent need to express the directory information as XML data. This caters to the growing breed of applications that are not LDAP-aware yet require information exchange with a LDAP directory server.

Directory Services Mark-up Language (DSML) defines the XML representation of LDAP information and operations. The LDAP Data Interchange Format (LDIF) is used to convey directory information, or a set of changes to be applied to directory entries. The former is called Attribute Value Record and the latter is called Change Record.

Benefits of DSML

Using DSML with Oracle Internet Directory and Internet applications makes it easier to flexibly integrate data from disparate sources. Also, DSML enables applications that do not use LDAP to communicate with LDAP-based applications, easily operating on data generated by an Oracle Internet Directory client tool or accessing the directory through a firewall.

DSML is based on XML, which is optimized for delivery over the Web. Structured data in XML will be uniform and independent of application or vendors, thus making possible numerous new flat file type synchronization connectors. Once in XML format, the directory data can be made available in the middle tier and have more meaningful searches performed on it.

DSML Syntax

A DSML version 1 document describes either directory entries, a directory schema or both. Each directory entry has a unique name called a distinguished name (DN). A directory entry has a number of property-value pairs called directory attributes. Every directory entry is a member of a number of object classes. An entry's object classes

constrain the directory attributes the entry can take. Such constraints are described in a directory schema, which may be included in the same DSML document or may be in a separate document.

The following subsections briefly explain the top-level structure of DSML and how to represent the directory and schema entries.

Top-Level Structure

The top-level document element of DSML is of the type `dsml`, which may have child elements of the following types:

```
directory-entries
directory-schema
```

The child element directory-entries may in turn have child elements of the type entry. Similarly the child element directory-schema may in turn have child elements of the types class and attribute-type.

At the top level, the structure of a DSML document looks like this:

```
<!- a document with directory & schema entries -->
  <dsml:directory-entries>
    <dsml:entry dn="...">...</dsml:entry>
    .
    .
    .
  </dsml:directory-entries>
  .
  .
  .
  <dsml:directory-schema>
    <dsml:class id="..." ...>...</dsml:class>
    <dsml:attribute-type id="..." ...>...</dsml:attribute-type>
    .
    .
    .
  </dsml:directory-schema>
</dsml:dsml>
```

Directory Entries

The element type `entry` represents a directory entry in a DSML document. The `entry` element contains elements representing the entry's directory attributes. The distinguished name of the entry is indicated by the XML attribute `dn`.

Here is an XML entry to describe the directory entry:

```
<dsml:entry dn="uid=Heman, c=in, dc=oracle, dc=com">
<dsml:objectclass>
  <dsml:oc-value>top</dsml:oc-value>
  <dsml:oc-value ref="#person">person</dsml:oc-value>
  <dsml:oc-value>organizationalPerson</dsml:oc-value>
  <dsml:oc-value>inetOrgPerson</dsml:oc-value>
</dsml:objectclass>
<dsml:attr name="sn">
<dsml:value>Siva</dsml:value></dsml:attr>
<dsml:attr name="uid">
<dsml:value>Heman</dsml:value></dsml:attr>
<dsml:attr name="mail">
```

```
<dsml:attr name="givenname">
<dsml:value>Siva V. Kumar</dsml:value></dsml:attr>
<dsml:attr name="cn">
<dsml:value>Svenugop@Oracle.com</dsml:value></dsml:attr>
<dsml:value>Siva Kumar</dsml:value></dsml:attr>
```

The `oc-value's ref` is a URI Reference to a class element that defines the object class. In this case it is a URI [9] Reference to the element that defines the `person` object class. The child elements `objectclass` and `attr` are used to specify the object classes and the attributes of a directory entry.

Schema Entries

The element type `class` represents a schema entry in a DSML document. The `class` element takes an XML attribute `id` to make referencing easier.

For example, the object class definition for the `person` object class might look like the following:

```
<dsml:class id="person" superior="#top"  type="structural">
  <dsml:name>person</dsml:name>
  <dsml:description>...</dsml:description>
  <dsml:object-identifier>2.5.6.6</object-identifier>
  <dsml:attribute ref="#sn" required="true"/>
  <dsml:attribute ref="#cn" required="true"/>
  <dsml:attribute ref="#userPassword" required="false"/>
  <dsml:attribute ref="#telephoneNumber" required="false"/>
  <dsml:attribute ref="#seeAlso" required="false"/>
  <dsml:attribute ref="#description" required="false"/>
</dsml:class>
```

The directory attributes are described in a similar way. For example, the attribute definition for the `cn` attribute may look like this:

```
<dsml:attribute-type id="cn">
  <dsml:name>cn</dsml:name>
  <dsml:description>...</dsml:description>
  <dsml:object-identifier>2.5.4.3</object-identifier>
  <dsml:syntax>1.3.6.1.4.1.1466.115.121.1.44</dsml:syntax>
</dsml:attribute-type>
```

Tools Enabled for DSML

With the XML framework, you can now use non-ldap applications to access directory data. The XML framework broadly defines the access points and provides the following tools:

- `ldapadd`
- `ldapaddmt`
- `ldapsearch`

> **See Also:** "Entry and Attribute Management Command-Line Tools Syntax" in Appendix A for information about syntax and usage.

The client tool `ldifwrite` generates directory data and schema LDIF files. If you convert these LDIF files to XML, you can store the XML file on an application server and query it. The query and response time is small compared to performing an LDAP operation against an LDAP server.

Glossary

access control item (ACI)

An attribute that determines who has what type of access to what directory data. It contains a set of rules for structural access items, which pertain to entries, and content access items, which pertain to attributes. Access to both structural and content access items may be granted to one or more users or groups.

access control list (ACL)

The group of access directives that you define. The directives grant levels of access to specific data for specific clients, or groups of clients, or both.

access control policy point

An entry that contains security directives that apply downward to all entries at lower positions in the **directory information tree (DIT)**.

ACI

See **access control item (ACI)**.

ACL

See **access control list (ACL)**.

ACP

See **access control policy point**.

administrative area

A subtree on a directory server whose entries are under the control (schema, ACL, and collective attributes) of a single administrative authority.

advanced symmetric replication (ASR)

See **Oracle9i Advanced Replication**

anonymous authentication

The process by which the directory authenticates a user without requiring a user name and password combination. Each anonymous user then exercises the privileges specified for anonymous users.

API

See **application program interface**.

application program interface

Programs to access the services of a specified application. For example, LDAP-enabled clients access directory information through programmatic calls available in the LDAP API.

ASR

See **Oracle9i Advanced Replication**

attribute

An item of information that describes some aspect of an entry. An entry comprises a set of attributes, each of which belongs to an **object class**. Moreover, each attribute has both a *type*, which describes the kind of information in the attribute, and a *value*, which contains the actual data.

attribute configuration file

In an Oracle Directory Integration Platform environment, a file that specifies attributes of interest in a connected directory.

attribute type

The kind of information an attribute contains, for example, `jobTitle`.

attribute uniqueness

An Oracle Internet Directory feature that ensures that no two specified attributes have the same value. It enables applications synchronizing with the enterprise directory to use attributes as unique keys.

attribute value

The particular occurrence of information appearing in that entry. For example, the value for the `jobTitle` attribute could be `manager`.

authentication

The process of verifying the identity of a user, device, or other entity in a computer system, often as a prerequisite to allowing access to resources in a system.

authorization

Permission given to a user, program, or process to access an object or set of objects.

binding

The process of authenticating to a directory.

central directory

In an Oracle Directory Integration Platform environment, the directory that acts as the central repository. In an Oracle Directory Integration and Provisioning platform environment, Oracle Internet Directory is the central directory.

certificate

An ITU x.509 v3 standard data structure that securely binds an identity to a public key. A certificate is created when an entity's public key is signed by a trusted identity: a **certificate authority (CA)**. This certificate ensures that the entity's information is correct and that the public key actually belongs to that entity.

certificate authority (CA)

A trusted third party that certifies that other entities—users, databases, administrators, clients, servers—are who they say they are. The certificate authority verifies the user's identity and grants a certificate, signing it with the certificate authority's private key.

certificate chain

An ordered list of certificates containing an end-user or subscriber certificate and its certificate authority certificates.

change logs

A database that records changes made to a directory server.

cipher suite

In SSL, a set of authentication, encryption, and data integrity algorithms used for exchanging messages between network nodes. During an SSL handshake, the two nodes negotiate to see which cipher suite they will use when transmitting messages back and forth.

cluster

A collection of interconnected usable whole computers that is used as a single computing resource. Hardware clusters provide high availability and scalability.

cold backup

The procedure to add a new **DSA** node to an existing replicating system by using the database copy procedure.

concurrency

The ability to handle multiple requests simultaneously. Threads and processes are examples of concurrency mechanisms.

concurrent clients

The total number of clients that have established a session with Oracle Internet Directory.

concurrent operations

The number of operations that are being executed on the directory from all of the concurrent clients. Note that this is not necessarily the same as the concurrent clients, because some of the clients may be keeping their sessions idle.

configset

See **configuration set entry**.

configuration set entry

A directory entry holding the configuration parameters for a specific instance of the directory server. Multiple configuration set entries can be stored and referenced at runtime. The configuration set entries are maintained in the subtree specified by the subConfigsubEntry attribute of the DSE, which itself resides in the associated **directory information base (DIB)** against which the servers are started.

connect descriptor

A specially formatted description of the destination for a network connection. A connect descriptor contains destination service and network route information.

The destination service is indicated by using its service name for Oracle9i release 9.2 database or its Oracle System Identifier (SID) for Oracle release 8.0 or version 7 databases. The network route provides, at a minimum, the location of the listener through use of a network address.

connected directory

In an Oracle Directory Integration Platform environment, an information repository requiring full synchronization of data between Oracle Internet Directory and itself—for example, an Oracle human Resources database.

consumer

A directory server that is the destination of replication updates. Sometimes called a slave.

contention

Competition for resources.

context prefix

The **DN** of the root of a **naming context**.

cryptography

The practice of encoding and decoding data, resulting in secure messages.

data integrity

The guarantee that the contents of the message received were not altered from the contents of the original message sent.

decryption

The process of converting the contents of an encrypted message (ciphertext) back into its original readable format (plaintext).

default knowledge reference

A **knowledge reference** that is returned when the base object is not in the directory, and the operation is performed in a naming context not held locally by the server. A default knowledge reference typically sends the user to a server that has more knowledge about the directory partitioning arrangement.

default identity management realm

In a hosted environment, one enterprise—for example, an application service provider—makes Oracle components available to multiple other enterprises and stores information for them. In such hosted environments, the enterprise performing the hosting is called the default identity management realm, and the enterprises that are hosted are each associated with their own identity management realm in the DIT.

default realm location

An attribute in the root Oracle Context that identifies the root of the default identity management realm.

delegated administrator

In a hosted environment, one enterprise—for example, an application service provider—makes Oracle components available to multiple other enterprises and stores information for them. In such an environment, a global administrator performs activities that span the entire directory. Other administrators—called delegated

administrators—may exercise roles in specific identity management realms, or for specific applications.

DES

Data Encryption Standard, a block cipher developed by IBM and the U.S. government in the 1970's as an official standard.

DIB

See **directory information base (DIB)**.

directory information base (DIB)

The complete set of all information held in the directory. The DIB consists of entries that are related to each other hierarchically in a **directory information tree (DIT)**.

directory information tree (DIT)

A hierarchical tree-like structure consisting of the DNs of the entries.

directory integration profile

In an Oracle Directory Integration Platform environment, an entry in Oracle Internet Directory that describes how Oracle Directory Integration and Provisioning platform communicates with external systems and what is communicated.

directory integration and provisioning server

In an Oracle Directory Integration Platform environment, the server that drives the synchronization of data between Oracle Internet Directory and a **connected directory**.

directory naming context

See **naming context**.

directory provisioning profile

A special kind of **directory integration profile** that describes the nature of provisioning-related notifications that the Oracle Directory Integration and Provisioning platform sends to the directory-enabled applications

directory replication group (DRG)

The directory servers participating in a replication agreement.

directory server instance

A discrete invocation of a directory server. Different invocations of a directory server, each started with the same or different configuration set entries and startup flags, are said to be different directory server instances.

directory-specific entry (DSE)

An entry specific to a directory server. Different directory servers may hold the same DIT name, but have different contents—that is, the contents can be specific to the directory holding it. A DSE is an entry with contents specific to the directory server holding it.

directory synchronization profile

A special kind of **directory integration profile** that describes how synchronization is carried out between Oracle Internet Directory and an external system.

directory system agent (DSA)

The X.500 term for a directory server.

distinguished name (DN)

The unique name of a directory entry. It comprises all of the individual names of the parent entries back to the root.

DIS

See **directory integration and provisioning server**

DIT

See **directory information tree (DIT)**

DN

See **distinguished name (DN)**

DRG

See **directory replication group (DRG)**

DSA

See **directory system agent (DSA)**

DSE

See **directory-specific entry (DSE)**

DSA-specific entries. Different DSAs may hold the same DIT name, but have different contents. That is, the contents can be specific to the DSA holding it. A DSE is an entry with contents specific to the DSA holding it.

encryption

The process of disguising the contents of a message and rendering it unreadable (ciphertext) to anyone but the intended recipient.

entry

The building block of a directory, it contains information about an object of interest to directory users.

export agent

In an Oracle Directory Integration Platform environment, an agent that exports data out of Oracle Internet Directory.

export data file

In an Oracle Directory Integration Platform environment, the file that contains data exported by an **export agent**.

export file

See **export data file**.

external agent

A directory integration agent that is independent of Oracle directory integration and provisioning server. The Oracle directory integration and provisioning server does not provide scheduling, mapping, or error handling services for it. An external agent is

typically used when a third party metadirectory solution is integrated with the Oracle Directory Integration Platform.

failover

The process of failure recognition and recovery. In an Oracle Application Server Cold Failover Cluster, an application running on one cluster node is transparently migrated to another cluster node. During this migration, clients accessing the service on the cluster see a momentary outage and may need to reconnect once the failover is complete.

fan-out replication

Also called a point-to-point replication, a type of replication in which a supplier replicates directly to a consumer. That consumer can then replicate to one or more other consumers. The replication can be either full or partial.

filter

A method of qualifying data, usually data that you are seeking. Filters are always expressed as DNs, for example: `cn=susie smith,o=acme,c=us`.

global administrator

In a hosted environment, one enterprise—for example, an application service provider—makes Oracle components available to multiple other enterprises and stores information for them. In such an environment, a global administrator performs activities that span the entire directory.

global unique identifier (GUID)

An identifier generated by the system and inserted into an entry when the entry is added to the directory. In a multimaster replicated environment, the GUID, not the DN, uniquely identifies an entry. The GUID of an entry cannot be modified by a user.

grace login

A login occurring within the specified period before password expiration.

group search base

In the Oracle Internet Directory default DIT, the node in the identity management realm under which all the groups can be found.

guest user

One who is not an anonymous user, and, at the same time, does not have a specific user entry.

GUID

See **global unique identifier (GUID)**.

handshake

A protocol two computers use to initiate a communication session.

hash

A number generated from a string of text with an algorithm. The hash value is substantially smaller than the text itself. Hash numbers are used for security and for faster access to data.

identity management

The process by which the complete security lifecycle for network entities is managed in an organization. It typically refers to the management of an organization's application users, where steps in the security life cycle include account creation, suspension, privilege modification, and account deletion. The network entities managed may also include devices, processes, applications, or anything else that needs to interact in a networked environment. Entities managed by an identity management process may also include users outside of the organization, for example customers, trading partners, or Web services.

identity management realm

A collection of identities, all of which are governed by the same administrative policies. In an enterprise, all employees having access to the intranet may belong to one realm, while all external users who access the public applications of the enterprise may belong to another realm. An identity management realm is represented in the directory by a specific entry with a special object class associated with it.

identity management realm-specific Oracle Context

An Oracle Context contained in each identity management realm. It stores the following information:

- User naming policy of the identity management realm—that is, how users are named and located
- Mandatory authentication attributes
- Location of groups in the identity management realm
- Privilege assignments for the identity management realm—for example: who has privileges to add more users to the Realm.
- Application specific data for that Realm including authorizations

import agent

In an Oracle Directory Integration Platform environment, an agent that imports data into Oracle Internet Directory.

import data file

In an Oracle Directory Integration Platform environment, the file containing the data imported by an **import agent**.

inherit

When an object class has been derived from another class, it also derives, or inherits, many of the characteristics of that other class. Similarly, an attribute subtype inherits the characteristics of its supertype.

instance

See **directory server instance**.

integrity

The guarantee that the contents of the message received were not altered from the contents of the original message sent.

Internet Engineering Task Force (IETF)

The principal body engaged in the development of new Internet standard specifications. It is an international community of network designers, operators,

vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet.

Internet Message Access Protocol (IMAP)

A protocol allowing a client to access and manipulate electronic mail messages on a server. It permits manipulation of remote message folders, also called mailboxes, in a way that is functionally equivalent to local mailboxes.

key

A string of bits used widely in cryptography, allowing people to encrypt and decrypt data; a key can be used to perform other mathematical operations as well. Given a cipher, a key determines the mapping of the plaintext to the ciphertext.

key pair

A **public key** and its associated **private key**.

See **public/private key pair**.

knowledge reference

The access information (name and address) for a remote **DSA** and the name of the **DIT** subtree that the remote DSA holds. Knowledge references are also called referrals.

latency

The time a client has to wait for a given directory operation to complete. Latency can be defined as wasted time. In networking discussions, latency is defined as the travel time of a packet from source to destination.

LDAP

See **Lightweight Directory Access Protocol (LDAP)**.

LDIF

See **LDAP Data Interchange Format (LDIF)**.

Lightweight Directory Access Protocol (LDAP)

A standard, extensible directory access protocol. It is a common language that LDAP clients and servers use to communicate. The framework of design conventions supporting industry-standard directory products, such as the Oracle Internet Directory.

LDAP Data Interchange Format (LDIF)

The set of standards for formatting an input file for any of the LDAP command-line utilities.

logical host

In an Oracle Application Server Cold Failover Cluster, one or more disk groups and pairs of host names and IP addresses. It is mapped to a physical host in the cluster. This physical host impersonates the host name and IP address of the logical host

man-in-the-middle

A security attack characterized by the third-party, surreptitious interception of a message. The third-party, the *man-in-the-middle*, decrypts the message, re-encrypts it (with or without alteration of the original message), and retransmits it to the originally-intended recipient—all without the knowledge of the legitimate sender and receiver. This type of security attack works only in the absence of **authentication**.

mapping rules file

In an Oracle Directory Integration Platform environment, the file that specifies mappings between Oracle Internet Directory attributes and those in a **connected directory**.

master definition site (MDS)

In replication, a master definition site is the Oracle Internet Directory database from which the administrator runs the configuration scripts.

master site

In replication, a master site is any site other than the master definition site that participates in LDAP replication.

matching rule

In a search or compare operation, determines equality between the attribute value sought and the attribute value stored. For example, matching rules associated with the `telephoneNumber` attribute could cause "(650) 123-4567" to be matched with either "(650) 123-4567" or "6501234567" or both. When you create an attribute, you associate a matching rule with it.

MD4

A one-way hash function that produces a 128-bit hash, or message digest. If as little as a single bit value in the file is modified, the MD4 checksum for the file will change. Forgery of a file in a way that will cause MD4 to generate the same result as that for the original file is considered extremely difficult.

MD5

An improved version of MD4.

MDS

See **master definition site (MDS)**

metadirectory

A directory solution that shares information between all enterprise directories, integrating them into one virtual directory. It centralizes administration, thereby reducing administrative costs. It synchronizes data between directories, thereby ensuring that it is consistent and up-to-date across the enterprise.

MTS

See **shared server**

multimaster replication

Also called peer-to-peer or *n*-way replication, a type of replication that enables multiple sites, acting as equals, to manage groups of replicated data. In a multimaster replication environment, each node is both a supplier and a consumer node, and the entire directory is replicated on each node.

naming attribute

The attribute used to compose the RDN of a new user entry created through Oracle Delegated Administration Services or Oracle Internet Directory Java APIs. The default value for this is `cn`.

naming context

A subtree that resides entirely on one server. It must be contiguous, that is, it must begin at an entry that serves as the top of the subtree, and extend downward to either leaf entries or **knowledge reference**s (also called referrals) to subordinate naming contexts. It can range in size from a single entry to the entire DIT.

native agent

In an Oracle Directory Integration Platform environment, an agent that runs under the control of the **directory integration and provisioning server**. It is in contrast to an **external agent**.

net service name

A simple name for a service that resolves to a connect descriptor. Users initiate a connect request by passing a user name and password along with a net service name in a connect string for the service to which they wish to connect:

```
CONNECT username/password@net_service_name
```

Depending on your needs, net service names can be stored in a variety of places, including:

- Local configuration file, `tnsnames.ora`, on each client
- Directory server
- Oracle Names server
- External naming service, such as NDS, NIS or CDS

nickname attribute

The attribute used to uniquely identify a user in the entire directory. The default value for this is `uid`. Applications use this to resolve a simple user name to the complete distinguished name. The user nickname attribute cannot be multi-valued—that is, a given user cannot have multiple nicknames stored under the same attribute name.

object class

A named group of attributes. When you want to assign attributes to an entry, you do so by assigning to that entry the object classes that hold those attributes.

All objects associated with the same object class share the same attributes.

OEM

See **Oracle Enterprise Manager**.

OID Control Utility

A command-line tool for issuing run-server and stop-server commands. The commands are interpreted and executed by the **OID Monitor** process.

OID Database Password Utility

The utility used to change the password with which Oracle Internet Directory connects to an Oracle database.

OID Monitor

The Oracle Internet Directory component that initiates, monitors, and terminates the Oracle directory server processes. It also controls the replication server if one is installed, and Oracle directory integration and provisioning server.

one-way function

A function that is easy to compute in one direction but quite difficult to reverse compute, that is, to compute in the opposite direction.

one-way hash function

A **one-way function** that takes a variable sized input and creates a fixed size output.

Oracle Call Interface (OCI)

An application programming interface (API) that enables you to create applications that use the native procedures or function calls of a third-generation language to access an Oracle database server and control all phases of SQL statement execution.

Oracle Delegated Administration Services

A set of individual, pre-defined services—called Oracle Delegated Administration Services units—for performing directory operations on behalf of a user. Oracle Internet Directory Self-Service Console makes it easier to develop and deploy administration solutions for both Oracle and third-party applications that use Oracle Internet Directory.

Oracle Directory Integration Platform

A component of **Oracle Internet Directory**. It is a framework developed to integrate applications around a central LDAP directory like Oracle Internet Directory.

Oracle directory integration and provisioning server

In an Oracle Directory Integration Platform environment, a daemon process that monitors Oracle Internet Directory for change events and takes action based on the information present in the **directory integration profile**.

Oracle Directory Manager

A Java-based tool with a graphical user interface for administering Oracle Internet Directory.

Oracle Enterprise Manager

A separate Oracle product that combines a graphical console, agents, common services, and tools to provide an integrated and comprehensive systems management platform for managing Oracle products.

Oracle Identity Management

An infrastructure enabling deployments to manage centrally and securely all enterprise identities and their access to various applications in the enterprise.

Oracle Internet Directory

A general purpose directory service that enables retrieval of information about dispersed users and network resources. It combines Lightweight Directory Access Protocol (LDAP) Version 3 with the high performance, scalability, robustness, and availability of Oracle9i.

Oracle Net Services

The foundation of the Oracle family of networking products, allowing services and their client applications to reside on different computers and communicate. The main function of Oracle Net Services is to establish network sessions and transfer data between a client application and a server. Oracle Net Services is located on each

computer in the network. Once a network session is established, Oracle Net Services acts as a data courier for the client and the server.

Oracle PKI certificate usages

Defines Oracle application types that a **certificate** supports.

Oracle Wallet Manager

A Java-based application that security administrators use to manage public-key security credentials on clients and servers.

See Also: *Oracle Advanced Security Administrator's Guide*

Oracle9i Advanced Replication

A feature in Oracle9i that enables database tables to be kept synchronized across two Oracle databases.

other information repository

In an Oracle Directory Integration and Provisioning platform environment, in which Oracle Internet Directory serves as the **central directory**, any information repository except Oracle Internet Directory.

partition

A unique, non-overlapping directory naming context that is stored on one directory server.

peer-to-peer replication

Also called multimaster replication or *n*-way replication. A type of replication that enables multiple sites, acting as equals, to manage groups of replicated data. In such a replication environment, each node is both a supplier and a consumer node, and the entire directory is replicated on each node.

PKCS #12

A **public-key encryption** standard (PKCS). RSA Data Security, Inc. PKCS #12 is an industry standard for storing and transferring personal authentication credentials—typically in a format called a **wallet**.

plaintext

Message text that has not been encrypted.

point-to-point replication

Also called fan-out replication is a type of replication in which a supplier replicates directly to a consumer. That consumer can then replicate to one or more other consumers. The replication can be either full or partial.

primary node

In an Oracle Application Server Cold Failover Cluster, the cluster node on which the application runs at any given time.

private key

In public-key cryptography, this key is the secret key. It is primarily used for decryption but is also used for encryption with digital signatures.

provisioning agent

An application or process that translates Oracle-specific provisioning events to external or third-party application-specific events.

provisioned applications

Applications in an environment where user and group information is centralized in Oracle Internet Directory. These applications are typically interested in changes to that information in Oracle Internet Directory.

profile

See **directory integration profile**

proxy user

A kind of user typically employed in an environment with a middle tier such as a firewall. In such an environment, the end user authenticates to the middle tier. The middle tier then logs into the directory on the end user's behalf. A proxy user has the privilege to switch identities and, once it has logged into the directory, switches to the end user's identity. It then performs operations on the end user's behalf, using the authorization appropriate to that particular end user.

public key

In public-key cryptography this key is made public to all, it is primarily used for encryption but can be used for verifying signatures.

public-key cryptography

Cryptography based on methods involving a public key and a private key.

public-key encryption

The process in which the sender of a message encrypts the message with the public key of the recipient. Upon delivery, the message is decrypted by the recipient using the recipient's private key.

public/private key pair

A mathematically related set of two numbers where one is called the private key and the other is called the public key. Public keys are typically made widely available, while private keys are available only to their owners. Data encrypted with a public key can only be decrypted with its associated private key and vice versa. Data encrypted with a public key cannot be decrypted with the same public key.

realm search base

An attribute in the root Oracle Context that identifies the entry in the DIT that contains all identity management realms. This attribute is used when mapping a simple realm name to the corresponding entry in the directory.

referral

Information that a directory server provides to a client and which points to other servers the client must contact to find the information it is requesting.

See also **knowledge reference**.

relational database

A structured collection of data that stores data in tables consisting of one or more rows, each containing the same set of columns. Oracle makes it very easy to link the data in multiple tables. This is what makes Oracle a relational database management

system, or RDBMS. It stores data in two or more tables and enables you to define relationships between the tables. The link is based on one or more fields common to both tables.

replica

Each copy of a naming context that is contained within a single server.

RDN

See **relative distinguished name (RDN)**.

registry entry

An entry containing runtime information associated with invocations of Oracle directory servers, called a **directory server instance**. Registry entries are stored in the directory itself, and remain there until the corresponding directory server instance stops.

relative distinguished name (RDN)

The local, most granular level entry name. It has no other qualifying entry names that would serve to uniquely address the entry. In the example, `cn=Smith,o=acme,c=US`, the RDN is `cn=Smith`.

remote master site (RMS)

In a replicated environment, any site, other than the **master definition site (MDS)**, that participates in Oracle9i Advanced Replication.

replication agreement

A special directory entry that represents the replication relationship among the directory servers in a **directory replication group (DRG)**.

response time

The time between the submission of a request and the completion of the response.

root DSE

See **root directory specific entry**.

root directory specific entry

An entry storing operational information about the directory. The information is stored in a number of attributes.

Root Oracle Context

In the Oracle Identity Management infrastructure, the Root Oracle Context is an entry in Oracle Internet Directory containing a pointer to the default identity management realm in the infrastructure. It also contains information on how to locate an identity management realm given a simple name of the realm.

SASL

See **Simple Authentication and Security Layer (SASL)**

scalability

The ability of a system to provide throughput in proportion to, and limited only by, available hardware resources.

schema

The collection of attributes, object classes, and their corresponding matching rules.

secondary node

In an Oracle Application Server Cold Failover Cluster, the cluster node to which an application is moved during a failover.

Secure Hash Algorithm (SHA)

An algorithm that takes a message of less than 264 bits in length and produces a 160-bit message digest. The algorithm is slightly slower than MD5, but the larger message digest makes it more secure against brute-force collision and inversion attacks.

Secure Socket Layer (SSL)

An industry standard protocol designed by Netscape Communications Corporation for securing network connections. SSL provides authentication, encryption, and data integrity using public key infrastructure (PKI).

service time

The time between the initiation of a request and the completion of the response to the request.

session key

A key for symmetric-key cryptosystems that is used for the duration of one message or communication session.

SGA

See **System Global Area (SGA)**.

SHA

See **Secure Hash Algorithm (SHA)**.

shared server

A server that is configured to allow many user processes to share very few server processes, so the number of users that can be supported is increased. With shared server configuration, many user processes connect to a dispatcher. The dispatcher directs multiple incoming network session requests to a common queue. An idle shared server process from a shared pool of server processes picks up a request from the queue. This means a small pool of server processes can server a large amount of clients. Contrast with dedicated server.

sibling

An entry that has the same parent as one or more other entries.

simple authentication

The process by which the client identifies itself to the server by means of a DN and a password which are not encrypted when sent over the network. In the simple authentication option, the server verifies that the DN and password sent by the client match the DN and password stored in the directory.

Simple Authentication and Security Layer (SASL)

A method for adding authentication support to connection-based protocols. To use this specification, a protocol includes a command for identifying and authenticating a user

to a server and for optionally negotiating a security layer for subsequent protocol interactions. The command has a required argument identifying a SASL mechanism.

single key-pair wallet

A **PKCS #12**-format **wallet** that contains a single user **certificate** and its associated **private key**. The **public key** is imbedded in the certificate.

slave

See **consumer**.

SLAPD

Standalone LDAP daemon.

smart knowledge reference

A **knowledge reference** that is returned when the knowledge reference entry is in the scope of the search. It points the user to the server that stores the requested information.

specific administrative area

Administrative areas control:

- Subschema administration
- Access control administration
- Collective attribute administration

A *specific* administrative area controls one of these aspects of administration. A specific administrative area is part of an autonomous administrative area.

sponsor node

In replication, the node that is used to provide initial data to a new node.

SSL

See **Secure Socket Layer (SSL)**.

subACLSubentry

A specific type of subentry that contains ACL information.

subclass

An object class derived from another object class. The object class from which it is derived is called its **superclass**.

subentry

A type of entry containing information applicable to a group of entries in a subtree. The information can be of these types:

- Access control policy points
- Schema rules
- Collective attributes

Subentries are located immediately below the root of an administrative area.

subordinate reference

A knowledge reference pointing downward in the DIT to a naming context that starts immediately below an entry.

subschema DN

The list of DIT areas having independent schema definitions.

subSchemaSubentry

A specific type of **subentry** containing schema information.

subtype

An attribute with one or more options, in contrast to that same attribute without the options. For example, a `commonName` (`cn`) attribute with American English as an option is a subtype of the `commonName` (`cn`) attribute without that option. Conversely, the `commonName` (`cn`) attribute without an option is the **supertype** of the same attribute with an option.

super user

A special directory administrator who typically has full access to directory information.

superclass

The object class from which another object class is derived. For example, the object class `person` is the superclass of the object class `organizationalPerson`. The latter, namely, `organizationalPerson`, is a **subclass** of `person` and inherits the attributes contained in `person`.

superior reference

A knowledge reference pointing upward to a DSA that holds a naming context higher in the DIT than all the naming contexts held by the referencing DSA.

supertype

An attribute without options, in contrast to the same attribute with one or more options. For example, the `commonName` (`cn`) attribute without an option is the supertype of the same attribute with an option. Conversely, a `commonName` (`cn`) attribute with American English as an option is a **subtype** of the `commonName` (`cn`) attribute without that option.

supplier

In replication, the server that holds the master copy of the naming context. It supplies updates from the master copy to the **consumer** server.

System Global Area (SGA)

A group of shared memory structures that contain data and control information for one Oracle database instance. If multiple users are concurrently connected to the same instance, the data in the instance SGA is shared among the users. Consequently, the SGA is sometimes referred to as the "shared global area." The combination of the background processes and memory buffers is called an Oracle instance.

system operational attribute

An attribute holding information that pertains to the operation of the directory itself. Some operational information is specified by the directory to control the server, for example, the time stamp for an entry. Other operational information, such as access

information, is defined by administrators and is used by the directory program in its processing.

TLS

See **Transport Layer Security (TLS)**

think time

The time the user is not engaged in actual use of the processor.

throughput

The number of requests processed by Oracle Internet Directory for each unit of time. This is typically represented as "operations per second."

Transport Layer Security (TLS)

A protocol providing communications privacy over the Internet. The protocol enables client/server applications to communicate in a way that prevents eavesdropping, tampering, or message forgery.

trusted certificate

A third party identity that is qualified with a level of trust. The trust is used when an identity is being validated as the entity it claims to be. Typically, the certificate authorities you trust issue user certificates.

trustpoint

See **trusted certificate**.

UTF-16

16-bit encoding of **Unicode**.The Latin-1 characters are the first 256 code points in this standard.

Unicode

A type of universal character set, a collection of 64K characters encoded in a 16-bit space. It encodes nearly every character in just about every existing character set standard, covering most written scripts used in the world. It is owned and defined by Unicode Inc. Unicode is canonical encoding which means its value can be passed around in different locales. But it does not guarantee a round-trip conversion between it and every Oracle character set without information loss.

UNIX Crypt

The UNIX encryption algorithm.

user search base

In the Oracle Internet Directory default DIT, the node in the identity management realm under which all the users are placed.

UTC (Coordinated Universal Time)

The standard time common to every place in the world. Formerly and still widely called Greenwich Mean Time (GMT) and also World Time, UTC nominally reflects the mean solar time along the Earth's prime meridian. UTC is indicated by a z at the end of the value, for example, 200011281010z.

UTF-8

A variable-width 8-bit encoding of **Unicode** that uses sequences of 1, 2, 3, or 4 bytes for each character. Characters from 0-127 (the 7-bit ASCII characters) are encoded with one byte, characters from 128-2047 require two bytes, characters from 2048-65535 require three bytes, and characters beyond 65535 require four bytes. The Oracle character set name for this is AL32UTF8 (for the Unicode 3.1 standard).

virtual host name

In an Oracle Application Server Cold Failover Cluster, the host name corresponding to this virtual IP address.

virtual IP address

In an Oracle Application Server Cold Failover Cluster, each physical node has its own physical IP address and physical host name. To present a single system image to the outside world, the cluster uses a dynamic IP address that can be moved to any physical node in the cluster. This is called the virtual IP address.

wallet

An abstraction used to store and manage security credentials for an individual entity. It implements the storage and retrieval of credentials for use with various cryptographic services. A wallet resource locator (WRL) provides all the necessary information to locate the wallet.

wait time

The time between the submission of the request and initiation of the response.

X.509

A popular format from ISO used to sign public keys.

Index

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W