Oracle® Internet Directory

Application Developer's Guide

10*g* (9.0.4)

Part No. B10461-01

September 2003

ORACLE®

Oracle Internet Directory Application Developer's Guide, 10*g* (9.0.4)

Part No. B10461-01

Primary Author: Richard Smith

Contributing Author: Jennifer Polk

Contributors: Ramakrishna Bollu, Saheli Dey, Bruce Ernst, Rajinder Gupta, Ashish Kolli, Stephen Lee, David Lin, Radhika Moolky, David Saslav

Graphic Artist: Valarie Moore

must be followed, or as required to translate it into languages other than English."

RSA and RC4 are trademarks of RSA Data Security. Portions of Oracle Internet Directory have been licensed by Oracle Corporation from RSA Data Security.

Oracle Directory Manager requires the Java™ Runtime Environment. The Java™ Runtime Environment, Version JRE 1.1.6. ("The Software") is developed by Sun Microsystems, Inc. 2550 Garcia Avenue, Mountain View, California 94043. Copyright (c) 1997 Sun Microsystems, Inc.

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

iPlanet is a registered trademark of Sun Microsystems, Inc.

Contents

4 Developing Provisioning-Integrated Applications

5 Developing Oracle Internet Directory Server Plug-ins

6 Developing Applications Integrated with Oracle Delegated Administration Services

Part II Oracle Internet Directory Programming Reference

7 The C API for Oracle Internet Directory

8 DBMS_LDAP PL/SQL Reference

9 DBMS_LDAP_UTL PL/SQL Reference

10 DAS_URL Interface Reference

11 Provisioning Integration API Reference

Part III Appendixes

A Syntax for LDIF and Command-Line Tools

B Sample Usage

C DSML Syntax

Glossary

Index

List of Figures

List of Tables

Send Us Your Comments

Oracle Internet Directory Application Developer's Guide, 10*g* (9.0.4)

Part No. B10461-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. 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?

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

- Electronic mail: appserverdocs@oracle.com
- FAX: (650) 506-7227 Attn: Server Technologies Documentation Manager
- Postal service:
  Oracle Corporation
  Server Technologies Documentation
  500 Oracle Parkway, Mailstop 4op11
  Redwood Shores, CA 94065
  USA

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

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

Preface

Oracle Internet Directory Application Developer's Guide provides information for enabling applications to access Oracle Internet Directory by using the C API and the PL/SQL API.

This preface contains these topics:

Audience

Oracle Internet Directory Application Developer's Guide is for application developers who wish to enable applications to store and update directory information in an Oracle Internet Directory server. It is also intended for anyone who wants to know how the Oracle Internet Directory C API, PL/SQL API, Java API, and Oracle extensions work.

Organization

Part I, Oracle Internet Directory and LDAP Programming Concepts

Chapter 1, "Introduction"

Briefly describes the intended audience and components of Oracle Internet Directory Software Developer's Kit 10*g* (9.0.4). It also lists the other components of Oracle Internet Directory and the platforms it supports.

Chapter 2, "Developing Applications with Standard LDAP APIs"

This chapter provides a brief overview of all of the major operations available in the C API and the PL/SQL API. It 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 LDAP APIs"

This chapter explains the concepts behind Oracle extensions to LDAP APIs. It 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"

This chapter 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 or third-party applications that are based on the Oracle platform.

Chapter 5, "Developing Oracle Internet Directory Server Plug-ins"

This chapter explains how to use the plug-in framework for the Oracle Internet Directory server to facilitate custom development.

Chapter 6, "Developing Applications Integrated with Oracle Delegated Administration Services"

This chapter explains how developers can use the DAS URL API to achieve integration with DAS.

Part II Oracle Internet Directory API Reference

Chapter 7, "The C API for Oracle Internet Directory"

Introduces the Oracle Internet Directory API and provides examples of how to use it

Chapter 8, "DBMS_LDAP PL/SQL Reference"

This chapter introduces the DBMS_LDAP package, which enables PL/SQL programmers to access data from LDAP servers. It provides examples of how to use DBMS_LDAP.

Chapter 9, "DBMS_LDAP_UTL PL/SQL Reference"

This chapter contains reference material for the DBMS_LDAP_UTL package, which contains Oracle Extension utility functions.

Chapter 10, "DAS_URL Interface Reference"

This chapter describes the Oracle extensions to the DAS_URL API.

Chapter 11, "Provisioning Integration API Reference"

This chapter 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, "Sample Usage"

This appendix provides sample code.

Appendix C, "DSML Syntax"

This appendix provides syntax and usage notes for DSML (XML) integration.

Glossary

Related Documentation

For more information, see these Oracle resources:

- Oracle9*i* Database Server and Oracle Application Server documentation sets, especially
  - *Oracle Internet Directory Administrator's Guide.*
  - *PL/SQL User's Guide and Reference*
  - *Oracle9i Application Developer's Guide - Fundamentals*
  - *Oracle Application Server 10g Security Guide*

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 username 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**.

Convention	Meaning	Example
Italics	Italic typeface indicates book titles or emphasis.	*Oracle9i 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 executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as 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 open 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 `U`*`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
[ ]	Brackets enclose one or more optional items. Do not enter the brackets.	DECIMAL (*digits* [, *precision*])
{ }	Braces enclose two or more items, one of which is required. Do not enter the braces.	{ENABLE | DISABLE}
|	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	{ENABLE | DISABLE} [COMPRESS | NOCOMPRESS]
...	Horizontal ellipsis points indicate either: ■ That we have omitted parts of the code that are not directly related to the example ■ That you can repeat a portion of the code	CREATE TABLE ... AS *subquery*; SELECT *col1*, *col2*, ... , *coln* FROM employees;
. . .	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	SQL> SELECT NAME FROM V$DATAFILE; NAME ---------- /fsl/dbs/tbs_01.dbf /fs1/dbs/tbs_02.dbf . . . /fsl/dbs/tbs_09.dbf 9 rows selected.
Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points 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. However, because these terms are not case sensitive, you can enter them in lowercase.	SELECT last_name, employee_id FROM employees; SELECT * FROM USER_TABLES; DROP TABLE hr.employees;

Convention	Meaning	Example

`lowercase`	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates 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 >	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 file name 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 scott/tiger TABLES=emp QUERY=\"WHERE job='SALESMAN' and sal<1600\"` `C:\>imp SYSTEM/`*`password`*` FROMUSER=scott TABLES=(emp, dept)`
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. For Windows NT, the default location 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`. 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\ora`*`nn`*, where *nn* is the latest release number. The Oracle home directory is located directly under *ORACLE_BASE*. All directory path examples in this guide follow OFA conventions. Refer to *Oracle9i Database Getting Started 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.

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 Corporation 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 Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.

What's New in Oracle Internet Directory Software Developer's Kit?

This section provides a brief description of new features introduced with the latest releases of the Oracle Internet Directory Software Developer's Kit, and points you to more information about each one.

New Features in Oracle Internet Directory Release 9.0.4

- Oracle Delegated Administration Services URL API

  This API enables you to build administrative and self-service consoles that can be used by delegated administrators and users to perform specified directory operations.

  See Also: Chapter 6, "Developing Applications Integrated with Oracle Delegated Administration Services"

- PL/SQL API Enhancements—These enhancements include:
  - New functions introduced in LDAP v3 standard. These were previously available in the core C-API, and are now made available through PL/SQL.
  - Functions to enable proxied access to middle-tier applications
  - Functions to create and manage provisioning profiles in the Oracle Directory Integration and Provisioning platform

See Also:

- Chapter 4, "Developing Provisioning-Integrated Applications"

- External authentication plug-in support—This feature enables administrators to use Microsoft Active Directory for storing and managing security credentials used by Oracle components.

  See Also: Chapter 5, "Developing Oracle Internet Directory Server Plug-ins"

- Server discovery using DNS—This feature enables Oracle Internet Directory clients to discover the host name and port number of the Oracle directory server running in a given enterprise. It reduces the administrative costs of maintaining Oracle Internet Directory clients in large deployments.

  See Also: "Server Discovery Functionality" on page 3-13

- Support for XML interface (DSML 1.0) OID SDK and tools—This feature enables LDAP tools to process XML as well as LDIF. APIs in Oracle Internet Directory can programmatically manipulate results and operations in DSML format.

  See Also: Link to relevant chapter or section for New_Feature_5

- Client side referral caching—This new feature enables clients to cache referral information and use it to speed up referral processing.

  See Also: "LDAP Session Handle Options" on page 7-10

Part I

Oracle Internet Directory Programming Concepts

Part I introduces the Oracle Internet Directory, summarizes the basic LDAP programming concepts, and explains how to directory-enable your applications. This part also includes short introductory chapters for each language-specific set of extensions.

It contains these chapters:

1

Introduction

This chapter briefly describes the intended audience and components of Oracle Internet Directory Software Developer's Kit 10*g* (9.0.4). It also lists the other components of Oracle Internet Directory and the platforms it supports.

This chapter contains these topics:

About Oracle Internet Directory Software Developer's Kit 10*g* (9.0.4)

Oracle Internet Directory SDK 10*g* (9.0.4) is intended for application developers using C, C++, and PL/SQL. Java developers can use the JNDI provider from Sun to access directory information in an Oracle Internet Directory server.

Components of the Oracle Internet Directory Software Developer's Kit

Oracle Internet Directory Software Developer's Kit 10*g* (9.0.4) consists of:

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

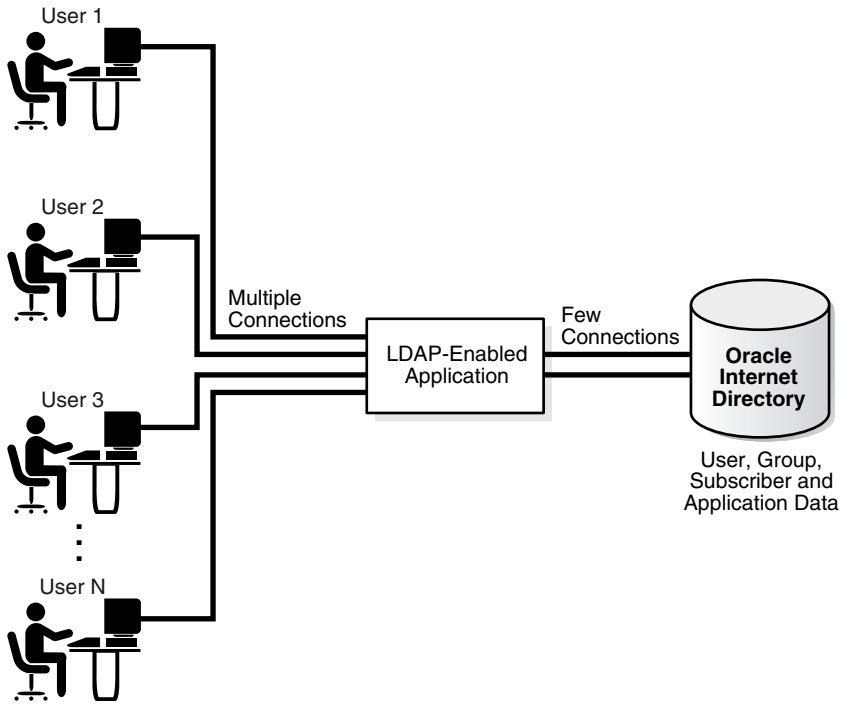
Application Development in the Oracle Internet Directory Environment

This section contains these topics:

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 in such environments.

Figure 1–1 A Directory-Enabled Application



As Figure 1–1 shows, when a user request needs an LDAP operation to be performed, the directory-enabled application performs the requested operation by using a smaller set of pre-created connections to Oracle Internet Directory.

Directory Interactions During Application Lifecycle

Table 1–1 gives an overview of the typical directory interactions that an application makes during its lifecycle..

Table 1–1 Interactions During Application Lifecycle

Point in Application Lifecycle	Logic
Application Installation	1. Create in Oracle Internet Directory an identity correspondent to the application. The application uses this identity to perform a majority of the LDAP operations. 2. Give this identity certain LDAP authorizations, by making it part of the correct LDAP groups, so that it can: Accept user credentials and authenticate them against Oracle Internet Directory Impersonate a user—that is, become a proxy user—if certain LDAP operations must be performed on behalf of the user
Application Startup and Bootstrap	The application must retrieve the credentials to authenticate itself to Oracle Internet Directory. If the application stores configuration metadata in Oracle Internet Directory, then 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.

Table 1–1 (Cont.) Interactions During Application Lifecycle

Point in Application Lifecycle	Logic
Application Runtime	For every end-user request that needs an LDAP operation, the application can: ■ Pick a connection from the pool of LDAP connections ■ Authenticate the end-user if required, and if Oracle Application Server Single Sign-On is not used ■ 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 regular API or the enhancements to it described in this chapter ■ Ensure that the effective user is now the application identity itself, once the operation is complete, if the application performed a proxy operation ■ 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 associated LDAP authorizations granted to the application identity.

Services and APIs for Integrating Applications with Oracle Internet Directory

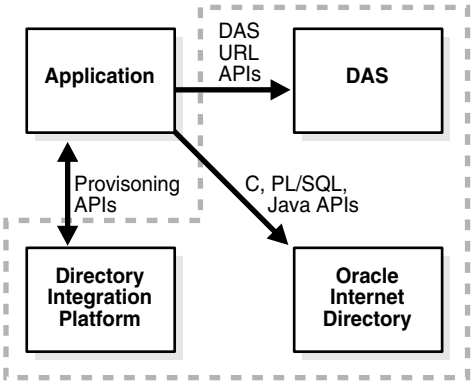
Application developers can integrate with Oracle Internet Directory by using the services and APIs listed and described in Table 1–2.

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 to be 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 additional programmatic interfaces that model various Identity Management related concepts.	Chapter 3, "Developing Applications with Oracle Extensions to the Standard LDAP APIs"
Oracle Delegated Administration Services	The Oracle Delegated Administration Services consist of a core self-service console and administrative interfaces that may be customized to support third-party applications.	Chapter 6, "Developing Applications Integrated with Oracle Delegated Administration Services" "Oracle Delegated Administration Services", in *Oracle Internet Directory Administrator's Guide*
Oracle Directory Provisioning Integration Service	You can use the Oracle Provisioning Integration System for provisioning third-party applications, as well as as a means of integrating other provisioning systems.	Chapter 4, "Developing Provisioning-Integrated Applications" "The Oracle Directory Provisioning Integration Service" in *Oracle Internet Directory Administrator's Guide*
Oracle Internet Directory Plug-ins	Oracle Internet Directory plug-ins can be used to customize the behavior of the directory server in certain deployment scenarios.	Chapter 5, "Developing Oracle Internet Directory Server Plug-ins" "Oracle Internet Directory Plug-In Framework" in *Oracle Internet Directory Administrator's Guide*

Figure 1–2 illustrates an application leveraging some of the services illustrated in Table 1–2 on page 1-6.

Figure 1–2 An Application Leveraging APIs and Services



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

- Through Oracle Internet Directory PL/SQL, C, or Java APIs, it performs LDAP operations directly against Oracle Internet Directory.
- For certain operations, it directs its users to some of the self-service capabilities of Oracle Delegated Administration Services.
- Through the Oracle Directory Provisioning Integration Service, it is notified of changes to certain user or group entries in Oracle Internet Directory.

Integrating Existing Applications with Oracle Internet Directory

Your enterprise may already have deployed certain applications to perform critical business applications. Table 1–3 lists and describes the services of the Oracle Internet Directory infrastructure that you can leverage to modify existing applications.

Table 1–3 Services for Modifying Existing Applications

Service	Description	More Information
Automated User Provisioning	You 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. When yo develop this agent, you must use the interfaces of the Oracle Directory Provisioning Integration Service.	Chapter 4, "Developing Provisioning-Integrated Applications."
User Authentication Services	If the user interface of the existing application is based on HTTP, then integrating it with Oracle HTTP Server and protecting its URL by using `mod_osso` authenticates all incoming user requests using the Oracle Application Server Single Sign-On.	*Oracle Application Server Single Sign-On Administrator's Guide*
Centralized User Profile Management	If the user interface of the existing application is based on HTTP, and it is integrated with Oracle Application Server Single Sign-On for authentication, then the application can leverage the Oracle Internet Directory Self-Service Console to enable centralized user profile management. The Self-Service Console can be customized by the deployment to address the specific needs of the application.	Chapter 6, "Developing Applications Integrated with Oracle Delegated Administration Services" "Oracle Delegated Administration Services", in *Oracle Internet Directory Administrator's Guide*

Integrating New Applications with Oracle Internet Directory

If you are developing a new application or planning a new release of an existing application, then you can leverage the services provided by the Oracle Internet Directory infrastructure extensively. Consider the integration points in described in Table 1–4 on page 1-9.

Table 1–4 Application Integration Points

Integration Point	Available Options	More Information
User Authentication Services	If the application is a J2EE based application, then it can use the services provided by the JAZN interface. If it relies on OC4J, then it can use the services provided by `mod_osso` to authenticate users and get important information about the user in the HTTP headers. If it is a stand-alone Web-based application, then it can still leverage Oracle Application Server Single Sign-On by becoming a partner application using the Oracle Application Server Single Sign-On APIs. Finally, if the application provides a non-Web based access interface, then it can authenticate users by using the Oracle Internet Directory LDAP APIs available in C, PL/SQL and Java.	*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", which contains reference sections for the various LDAP APIs

Table 1–4 (Cont.) Application Integration Points

Integration Point	Available Options	More Information
User Authorization Services	If the application is a J2EE-based application, then it can use the services provided by the JAZN interface to implement and enforce user authorizations to application defined resources. The application can model authorizations as groups in Oracle Internet Directory and then check the authorizations of a user by checking his or her group membership. It can do this by using the Oracle Internet Directory LDAP APIs available in C, PL/SQL and Java.	*Oracle Application Server Containers for J2EE User's Guide* Part II, "Oracle Internet Directory Programming Reference", which contains reference sections for the various LDAP APIs
Centralized Profile Management	You 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 is integrated with Oracle Application Server Single Sign-On for authentication, then the application can leverage the Oracle Internet Directory Self-Service Console to enable centralized user profile management. You can customize the Self-Service Console to address the specific needs of the application. The application can also retrieve these profiles at runtime by using the Oracle Internet Directory LDAP APIs available in C, PL/SQL and Java.	The chapter on deployment considerations in *Oracle Internet Directory Administrator's Guide* Chapter 6, "Developing Applications Integrated with Oracle Delegated Administration Services" "Oracle Delegated Administration Services", in *Oracle Internet Directory Administrator's Guide* Part II of this guide, which contains reference sections for the various LDAP APIs
Automated User Provisioning	If the user interface of the application is based on HTTP, and it is integrated with Oracle Application Server Single Sign-On for authentication, then you can implement automated user provisioning the very first time a user accesses the application. You can integrate the application in the Oracle Identity Management Infrastructure with the Oracle Directory Provisioning Integration Service. The application can then provision or deprovision user accounts automatically in response to such administrative actions as adding, modifying, or deleting an identity.	Chapter 4, "Developing Provisioning-Integrated Applications"

Other Components of Oracle Internet Directory

The following components of Oracle Internet Directory 10*g* (9.0.4), not part of the Oracle Internet Directory Software Developer's Kit, can be obtained separately:

- Oracle directory server, an LDAP Version 3-compliant directory server
- Oracle directory replication server
- Oracle Directory Manager, a Java-based graphical user interface
- Oracle Internet Directory bulk tools
- *Oracle Internet Directory Administrator's Guide*

Operating Systems Supported

Oracle Internet Directory servers and clients support these operating systems:

- HPUX (64 Bit) - 11.0 & 11i
- Linux (32 bit)—Red Hat AS 2.1 and United Linux 1.0
- AIX 5L (64 bit)—5.1 and 5.2
- HP Tru64—5.1b

2

Developing Applications with Standard LDAP APIs

This chapter provides a brief overview of all of the major operations available in the standard LDAP API. It provides developers a general understanding of **Lightweight Directory Access Protocol (LDAP)** and basic knowledge to integrate with the standard APIs.

This chapter contains these topics:

History of LDAP

LDAP began as a lightweight front end to the X.500 Directory Access Protocol. To simplify X.500 Directory Access Protocol, LDAP:

- Uses TCP/IP connections which are much more lightweight compared to the OSI communication stack required by X.500 implementations
- Eliminates little-used and redundant features found in the X.500 Directory Access Protocol
- Represents most data elements by using simple formats. These formats are easier to process than the more complicated and highly structured representations found in X.500.
- Encodes data for transport over networks by using a simplified version of the same encoding rules used by X.500

Overview of LDAP Models

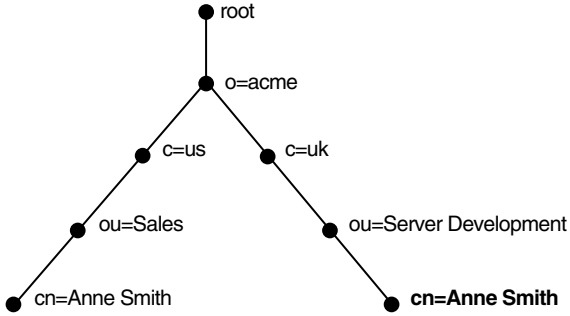
LDAP defines four basic models to describe its operations. This section contains these topics:

LDAP Naming Model

The LDAP naming model allows directory information to be referenced and organized. Each entry in a directory is uniquely identified by a **DN**. The distinguished name tells you exactly where the entry resides in the directory's hierarchy. This hierarchy is represented by a **directory information tree (DIT)**.

To understand the relation between a distinguished name and a directory information tree, look at the example in Figure 2–1.

Figure 2–1 A Directory Information Tree



The DIT in Figure 2–1 diagrammatically represents 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, while the other 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 is:

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

Note that the conventional format of a distinguished name places the lowest DIT component at the left, then follows it with the next highest component, thus moving progressively up to the root.

Within a distinguished name, the lowest component is called the **relative distinguished name (RDN)**. For example, in the above entry for Anne Smith, the RDN is `cn=Anne Smith`. Similarly, the RDN for the entry immediately above Anne Smith's RDN is `ou=Server Development`, 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. For example, within the global organization in Figure 2–1, to avoid confusion between the two

Anne Smiths, you would use each one's full DN. (If there are potentially two employees with the same name in the same organizational unit, you could use additional mechanisms, such as identifying each employee with a unique identification number.)

LDAP Information Model

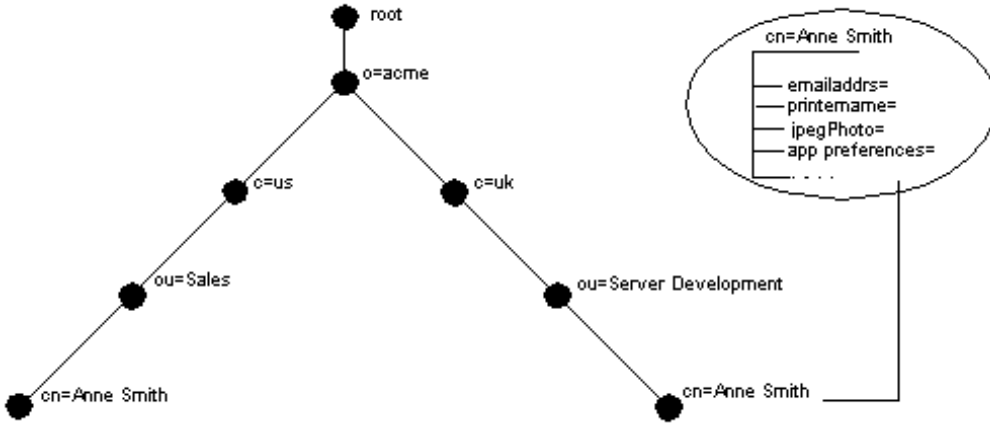
The LDAP information model determines the form and character of information in the directory. It is centered around entries, which are composed of attributes. In a directory, each collection of information about an object is called an **entry**. For example, a typical telephone directory includes entries for people, and a library card catalog contains entries for books. Similarly, an online directory might include entries for employees, conference rooms, e-commerce partners, or shared network resources such as printers.

In a typical telephone directory, an entry for a person contains such information items as an address and a phone number. In an online directory, such an information item is called an **attribute**. Attributes in a typical employee entry can include, for example, a job title, an e-mail address, or a phone number.

For example, in Figure 2–2, the entry for Anne Smith in Great Britain (uk) has several attributes, each providing specific information about her. These are listed in the balloon to the right of the tree, and they include `emailaddrs`, `printername`,

`jpegPhoto`, and `app preferences`. Moreover, each bullet in Figure 2–2 is also an entry with attributes, although the attributes for each 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—for example, `jobTitle`. The **attribute value** is the particular occurrence of information appearing in that entry. For example, the value for the `jobTitle` attribute could be `manager`.

LDAP Functional Model

The LDAP functional model determines what operations can be performed on the information. There are 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 for modifying the directory: Modify: change existing entries. It allows attributes and values to be added and deleted. 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, allowing a client to initiate a session and prove its identity to the directory. Several authentication methods are supported, from simple clear-text password to public key-based authentication. The unbind operation is used to terminate a directory session.

LDAP Security Model

The LDAP security model allows information in the directory to be secured.

This section contains these topics:

- 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 Protection: Ensuring protection of user passwords through any of four encryption options
- Password Policies: Enabling you to set rules that govern how passwords are used

Authentication

Authentication is the process by which the directory server establishes the true identity of the user connecting to the directory. It occurs when an LDAP session is established by means of the ldap-bind operation. Every session has an associated user identity, also referred to as an authorization ID.

To ensure that the identities of users, hosts, and clients are correctly known, Oracle Internet Directory provides three authentication options: anonymous, simple, and SSL.

Anonymous Authentication If your directory is available to everyone, then you can allow users to log in to the directory anonymously. When using **anonymous authentication**, users simply leave blank the user name and password fields when they log in. Each anonymous user then exercises whatever privileges are specified for anonymous users.

Simple Authentication In this case, 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.

Authentication Using Secure Sockets Layer (SSL) **Secure Socket Layer (SSL)** is an industry standard protocol for securing network connections. It provides authentication through the exchange of **certificates** that 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 the parties involved.

You can use SSL in one of three authentication modes:

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/decryption is 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. Both the client and server send certificates to each other.

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 the SSL port. (The default SSL port is 636.)
2. SSL performs the handshake between client and 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

Authorization is the process of ensuring that a user reads or updates only the information for which that user has privileges. When directory operations are attempted within a directory session, the directory server ensures that the user—identified by the authorization ID associated with the session—has the requisite permissions to perform those operations. Otherwise, the operation is disallowed. Through this mechanism, the directory server protects directory data from unauthorized operations by directory users. This mechanism is called access control.

An access control information item (ACI) is the directory metadata that captures the administrative policies relating to access control.

ACI is stored in Oracle Internet Directory as user-modifiable operational attributes. Typically, a list of these ACI attribute values, called an Access Control List (ACL), is

associated with directory objects. The attribute values on that list govern the access policies for those directory objects.

ACIs are represented and 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, that is, 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 ensures that data has not been modified, deleted, or replayed during transmission by using SSL. This SSL feature generates a cryptographically secure message digest—through cryptographic checksums using either the **MD5** algorithm or the **Secure Hash Algorithm (SHA)**—and includes it with each packet sent across the network.

Data Privacy

Oracle Internet Directory ensures that data is not disclosed during transmission by using **public-key encryption** available with Secure Sockets Layer (SSL). 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 the recipient's private key. Specifically, Oracle Internet Directory supports two levels of encryption available through SSL:

- DES40

  The DES40 algorithm, available internationally, is a variant of **DES** 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. This feature gives commercial customers a choice in the algorithm they use, regardless of their geographic location.

- RC4_40

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

Password Protection During installation, the protection scheme for passwords was set. You can change that initial configuration by using either Oracle Directory Manager or ldapmodify. You must be a superuser to change the type of password encryption.

To encrypt passwords, Oracle Internet Directory uses the **MD4** algorithm as the default. MD4 is a one-way hash function that produces a 128-bit hash, or message digest. You can change this default to one of the following:

- **MD5**—An improved, and more complex, version of MD4
- **SHA**—Secure Hash Algorithm, which produces a 160-bit hash, longer than MD5. The algorithm is slightly slower than MD5, but the larger message digest makes it more secure against brute-force collision and inversion attacks.
- **UNIX Crypt**—The UNIX encryption algorithm
- No Encryption

The value you specify is stored in the `orclCryptoScheme` attribute in the **root DSE**. This attribute is single-valued.

During authentication to a directory server, a user enters a password in clear text. The server then hashes the password by using the specified encryption algorithm, and verifies it against the hashed password in the `userPassword` attribute. If the hashed password values match, then the server authenticates the user. If the hashed password values do not match, then the server sends the user an Invalid Credentials error message.

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 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 own passwords

About Standard LDAP APIs

The standard LDAP enables you to perform the fundamental LDAP operations described in the previous section. The standard LDAP APIs are available in the these languages:

- C—Part of the Oracle Internet Directory Software Developer's Kit
- PL/SQL—Part of the Oracle Internet Directory Software Developer's Kit as DBMS_LDAP
- Java—Part of the Sun Microsystems JNDI package

All of these APIs use TCP/IP connections, are based on LDAP Version 3, and support SSL connectivity to Oracle Internet Directory.

This section contains these topics:

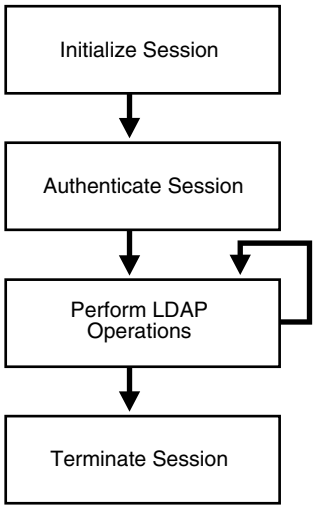
API Usage Model

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



Later sections in this chapter explain the important features of the API with respect to each of these steps.

About 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.9.0`.

> **See Also:** "Sample C API Usage" on page 7-65 for more details on how to use the SSL and non-SSL modes

About the Java API

Java developers can use the JNDI LDAP service provider from Sun Microsystems to access directory information in an Oracle Internet Directory server.

> **See Also:** `http://java.sun.com` for complete information about the JNDI provider from Sun

About the DBMS_LDAP Package

The DBMS_LDAP package enables PL/SQL applications to access data located in enterprise-wide LDAP servers. The naming and syntax of the function calls are similar to those of the Oracle Internet Directory C API functions and comply with the current recommendations from the **Internet Engineering Task Force (IETF)** for the LDAP C-API. However, the PL/SQL API contains only a subset of the functions available in the C API. In particular, only synchronous calls to the LDAP server are available in the PL/SQL API.

To use the PL/SQL LDAP API, load it into the database. You do this by using a script called `catldap.sql` that is located in the `$ORACLE_HOME/rdbms/admin` directory. You must be connected as SYSUSER using the SQL*Plus command line tool. You must also execute SQL*Plus in the ORACLE HOME in which your database is present.

The following is a sample command sequence that you can use to load the DBMS_LDAP package:

```
SQL> CONNECT / AS SYSDBA
SQL> @?/rdbms/admin/catldap.sql
```

Initializing an LDAP Session

All LDAP operations require clients to establish an LDAP session with the LDAP server. To perform LDAP operations, a database session must first initialize and open an LDAP session.

This section contains these topics:

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

Initializing the Session by Using the C API

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.

Syntax

```
LDAP *ldap_init
(
```

```
    const char      *hostname,
    int             portno
)
;
```

Parameters

Table 2–3 Parameters for ldap_init()

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 itself with a colon (:) character. The hosts will be tried in the order listed, stopping with the first one to which 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 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 then this parameter is ignored.

ldap_init() and ldap_open() both return a session handle, that 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 in which case the operating system error reporting mechanism can be checked to see why the call failed.

Initializing the Session by Using JNDI

See Also: The following URL `http://java.sun.com` for complete information about the JNDI provider from Sun

Initializing the Session by Using DBMS_LDAP

Initialization occurs by means of a call to the function `DBMS_LDAP.init()`. The function 'init' has the following syntax:

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

To establish an LDAP session, the function `init` requires a valid host name and a port number. It allocates a data structure for the LDAP session and returns a handle of the type `DBMS_LDAP.SESSION` to the caller. The handle returned from the call to `init` should be used in all subsequent LDAP operations with the API. The DBMS_LDAP API uses the LDAP 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. There is an upper limit of 64 simultaneous active LDAP connections. Typically, multiple LDAP sessions within the same database session are opened if:

- There is a requirement to get data from multiple LDAP servers simultaneously
- There is a requirement to have open sessions using multiple LDAP identities

---

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.

Authenticating an LDAP Session

Before initiating any of the LDAP operations, an individual or application seeking to perform operations against an LDAP server must be authenticated. If the `dn` and `passwd` parameters are `NULL`, then the LDAP server assigns a special identity, called `anonymous`, to the application. Typically, the `anonymous` identity is associated with the least privileges in an LDAP directory.

When a bind operation is completed, the directory server remembers the new identity until either another bind is done or the LDAP session is terminated by using `unbind_s`. The identity is used by the LDAP server to enforce the security model specified by the enterprise administration. In particular, this identity helps the LDAP server determine whether the user or application has sufficient privileges to perform search, update, or compare operations in the directory.

Note that the password for the bind operation is sent in the clear over the network. If the network is not secure, then consider using SSL for authentication as well as secure data transport for all LDAP operations.

This section contains these topics:

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

Authenticating an LDAP Session by Using the C API

The function ldap_simple_bind_s() enables applications to authenticate to the directory server by using certain credentials.

The function ldap_simple_bind_s() has the following syntax:

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

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 that identity.

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

Authenticating an LDAP Session by Using JNDI

There is no special function to perform authentication. The desired authentication parameters are set up at initialization time.

> **See Also:** The following URL `http://java.sun.com` for complete information about the JNDI provider from Sun

Authenticating an LDAP Session by Using DBMS_LDAP

The functions `simple_bind_s` enables applications to authenticate to the directory server by using certain credentials. The function `simple_bind_s` has the following syntax:

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

The function `simple_bind_s` requires the LDAP session handle obtained from `init` as the first parameter. It also requires an LDAP **distinguished name (DN)** of an entry. This DN represents the identity that the application uses when it authenticates

The following PL/SQL code snippet shows a typical usage of the initialization, authentication, and cleanup functions just described.

```
DECLARE
    retval       PLS_INTEGER;
    my_session   DBMS_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 to the LDAP server on the computer `yow.acme.com` that is listening for requests at TCP/IP port number 389. Then an authentication is performed with the identity of `cn=orcladmin` whose password is `welcome`. This authenticates the LDAP session and paves the way for regular LDAP operations.

Searching the Directory

Searches are the most frequently used LDAP operations. The LDAP search operation allows applications to select and retrieve entries from the directory by using complex search criteria.

This section contains these topics:

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

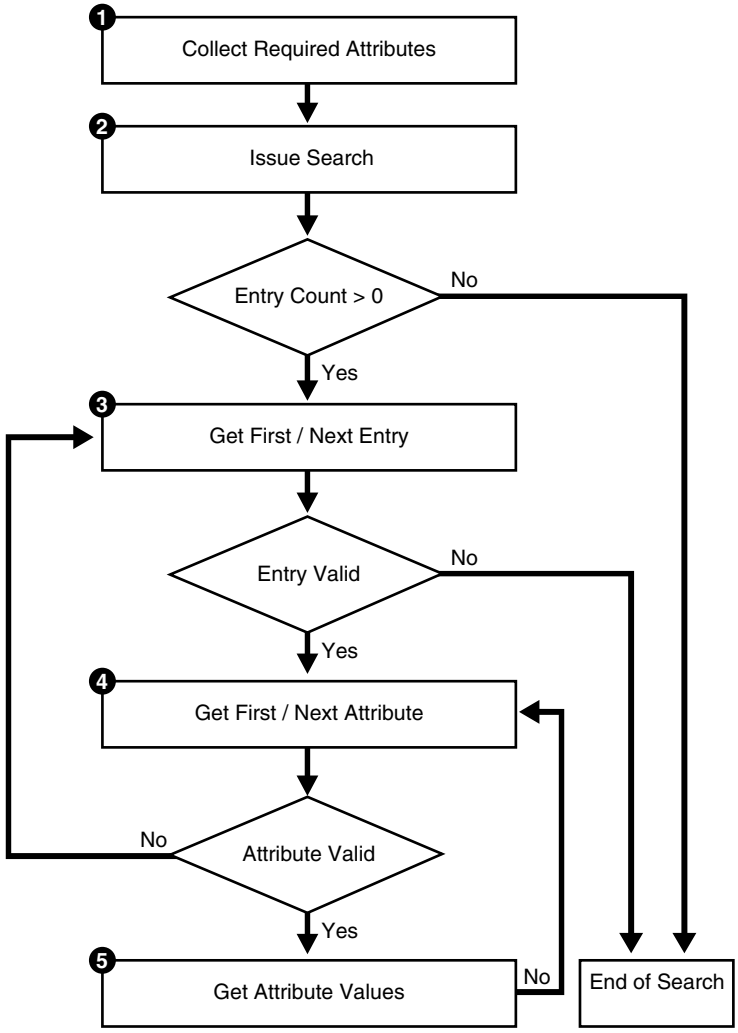
Flow of Search-Related Operations

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

1. Decide the attributes that need to be returned, and compose them into an array.
2. Initiate the search operation with the desired options and filters.
3. From the result set get an entry.
4. For the entry obtained in Step 3, get an attribute.
5. For the attribute obtained in Step 4, get all of the values and copy them 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 illustrates the above steps in more detail.

Figure 2–4 Flow of Search-Related Operations



Search Scope

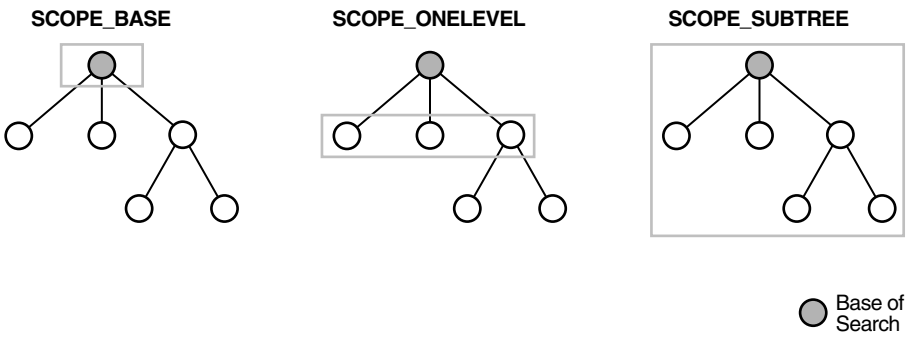
The scope of the search determines the number of entries relative to the base of the search that the directory server examines to see if they match the given filter condition. One of three options can be specified when invoking either `search_s()` or `search_st()` functions:

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 base of the search to see if it matches the given criteria in the filter.
SCOPE_ONELEVEL	The directory server looks only at all of the entries that are immediate children of the base object to see if they match the given criteria in the filter.
SCOPE_SUBTREE	The directory server looks at the entire LDAP subtree rooted at and including the base object.

Figure 2–5 illustrates the difference between the three scope options.

Figure 2–5 The Three Scope Options



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

Filters

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). This section provides a brief overview of the various options available for the filters.

There are six kinds of basic search filters that take an *attribute operator value* format. The following table summarizes the basic search filters:

Table 2–6 Search Filters

Filter Type	Format	Example	Matches
Equality	(*attr=value*)	(`sn=Keaton`)	Surnames exactly equal to Keaton.
Approximate	(*attr~=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 in “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.

The basic filters in Table 2–6 can be combined to form more complex filters using the Boolean operators and a prefix notation. The `&` character represents AND, the `|` character represents OR, and the `!` character represents NOT.

Table 2–7 summarizes the fundamental Boolean operations:

Table 2–7 Boolean Operators

Filter Type	Format	Example	Matches
AND	(&(<filter1>)(<filter2>)...)	`(&(sn=keaton)(objectclass=inetOrgPerson))`	Entries with surname of Keaton AND objectclass 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 shown above can themselves be combined to create arbitrarily complex nested filters.

Searching the Directory by Using the C API

The function `ldap_search-s()` can be used to initiate a synchronous search operation request in the directory.

The syntax for `ldap_search_s()` is:

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

Flow of Search Operation:

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

1. Decide the attributes that need to be returned, and compose them into an array of strings with the array being NULL terminated.
2. Initiate the search operation with the desired options and filters using ldap_search_s() function.
3. From the result-set get an entry using ldap_first_entry() or ldap_next_entry() function.
4. For the entry obtained in Step 3, get an attribute using ldap_first_attribute() or ldap_next_attribute() function.
5. For the attribute obtained in Step 4, get all of the values and copy them into local variables using ldap_get_values() or ldap_get_values_len().
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 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.
attrsonly	If set to 1, only returns the attributes.
res	The search results are returned in this argument.

Searching the Directory by Using JNDI

See Also: The following URL `http://java.sun.com` for complete information about the JNDI provider from Sun

Searching the Directory by Using DBMS_LDAP

The function available for initiating searches in the DBMS_LDAP API is `DBMS_LDAP.search_s()`.

The syntax for DBMS_LDAP.search_s() is:

```
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;
```

Both functions take the arguments listed and described in Table 2–9.

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

In addition to search_s, several support functions in the API help in retrieving search results. These are highlighted in the following section.

Flow of Search Operation:

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

1. Decide the attributes that need to be returned, and compose them into the DBMS_LDAP.STRING_COLLECTION data-type.
2. Initiate the search operation with the desired options and filters using DBMS_LDAP.search_s() or DBMS_LDAP.search_st().

3. From the result-set get an entry using `DBMS_LDAP.first_entry()` or `DBMS_LDAP.next_entry()` function.
4. For the entry obtained in Step 3, get an attribute using `DBMS_LDAP.first_attribute()` or `DBMS_LDAP.next_attribute()` function.
5. For the attribute obtained in Step 4, get all of the values and copy them into local variables using `DBMS_LDAP.get_values()` or `DBMS_LDAP.get_values_len()` function.
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

Once an LDAP session handle is obtained and all of the desired LDAP related work is complete, the LDAP session must be destroyed. This is accomplished through a call to ldap_unbind_s().

The function ldap_unbind_s() has the following syntax:

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

A successful call to `ldap_unbind_s()` function closes the TCP/IP connection to the directory server, de-allocates all the system resources consumed by the LDAP session and returns the integer `LDAP_SUCCESS` to its callers. Once the `ldap_unbind_s()` function is invoked on a particular session, no other LDAP operations on that session can succeed, unless a new LDAP session is initialized with a call to `ldap_init()`.

Terminating the Session by Using JNDI

See Also: The following URL `http://java.sun.com` for complete information about the JNDI provider from Sun

Terminating the Session by Using DBMS_LDAP

Once an LDAP session handle is obtained and all of the desired LDAP-related work is complete, the LDAP session must be destroyed. This is accomplished through a call to `DBMS_LDAP.unbind_s()`. The function `unbind_s` has the following syntax:

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

A successful call to `unbind_s` closes the TCP/IP connection to the LDAP server, de-allocates all system resources consumed by the LDAP session, and returns the integer `DBMS_LDAP.SUCCESS` to its callers. Once the `unbind_s` function is invoked on a particular session, no other LDAP operations on that session can succeed unless the session is re-initialized with a call to `init`.

3

Developing Applications with Oracle Extensions to the Standard LDAP APIs

This chapters explains the concepts behind Oracle extensions to LDAP APIs, and describes the abstract entities that are modeled by the extensions as well as the usage model of those extensions.

This chapter contains these topics:

Overview of Oracle Extensions to the Standard API

Based on the entities on which they operate, the functionalities provided by the API extensions can be categorized as follows:

- User management—This functionality enables applications to get or set various user related properties
- Group management—This functionality enables applications to query group properties
- Realm management—This functionality enables applications to get or set such identity management realm-related properties as user search base
- Server discovery management—This functionality enables applications to locate a directory server in the Domain Name System (DNS)
- SASL management—This functionality enables applications to authenticate to the directory by using SASL Digest-MD5 authentication

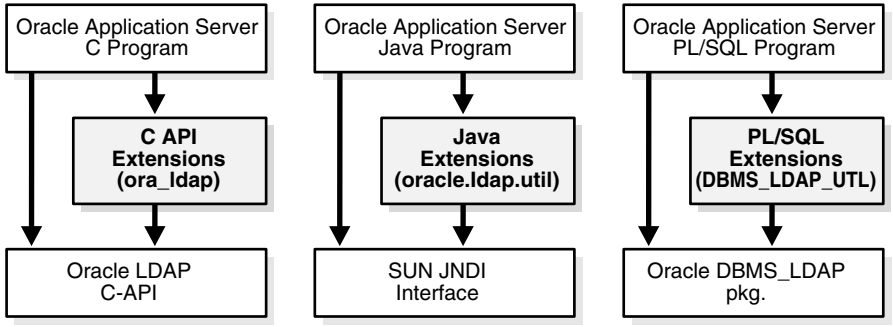
The primary users of the extensions described in this chapter are backend applications that must perform LDAP lookups for users, groups, applications, or hosted companies. This section describes how these applications integrate these API extensions into their logic—that is, the usage of the API extensions only. It contains these topics:

- Using the API Extensions in PL/SQL
- Using the API Extensions in Java
- Installation and First Use of Oracle Extensions to the Standard API

> **See Also:** "Architecture of a Directory-Enabled Application" on page 1-3 for a conceptual description of the usage model

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

Figure 3–1 Oracle API Extensions



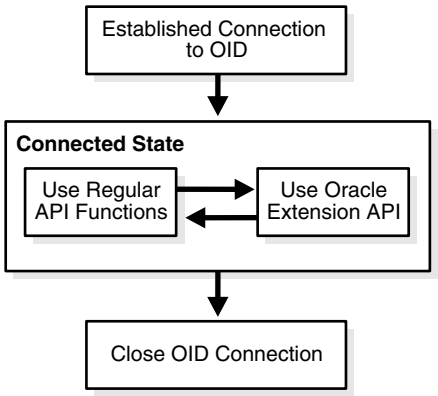
As Figure 3–1 shows, in the PL/SQL and Java languages, the API extensions are layered on top of existing APIs:

- Oracle's DBMS_LDAP PL/SQL API, for PL/SQL programs
- Sun's LDAP JNDI Service Provider, for Java programs
- Oracle's LDAP C API for C programs

Applications must access the underlying APIs for such common things as establishing and closing connections, and looking up directory entries not covered by the API extensions.

Figure 3–2 shows the programmatic flow of control for using the API extensions described in this chapter.

Figure 3–2 Programmatic Flow of API Extensions



As Figure 3–2 shows, the applications first establish a connection to Oracle Internet Directory. They can then use existing API functions and the API extensions interchangeably.

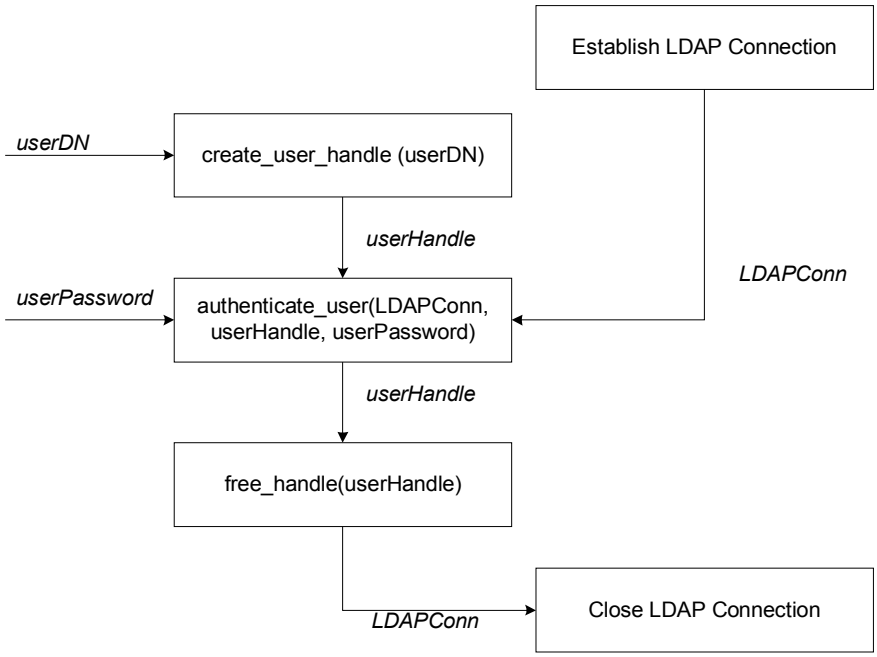
Using the API Extensions in PL/SQL

Most of the extensions described in this chapter provide helper functions to access data in relation to such specific LDAP entities as users, groups, realms, and applications. In many cases, you must pass a reference to one of these entities to the API functions. These API extensions use opaque data structures, called handles. For example, an application that needs to authenticate a user would follow these steps:

1. Establish an LDAP connection, or get it from a pool of connections.
2. Create a user handle based on user input. This could be a DN, or a GUID, or a simple Oracle Application Server Single Sign-On ID.
3. Authenticate the user with the LDAP connection handle, user handle, and credentials.
4. Free the user handle.
5. Close the LDAP connection, or return the connection back to the pool of connections.

Figure 3–3 illustrates this usage model.

Figure 3–3 Programming Abstractions for the PL/SQL Language



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

Instead of handles, LDAP entities—that is, users, groups, realms, and applications—are modeled as Java objects 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 `DirContext` object was retrieved from a pool of existing `DirContext` objects, return it to that pool.

Unlike C and PL/SQL, Java language usage does not need to explicitly free objects because the Java garbage collection mechanism can do it.

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 results. It is 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 `objectclass` values of each entry.

> **See Also:** "Java Sample Code" on page B-33 for more sample uses of the `PropertySetCollection`, `PropertySet`, and `Property` classes

Installation and First Use of Oracle Extensions to the Standard API

Table 3–1 provides information about installation and first use for each API.

Table 3–1 Information about Installation and First Use

Language	Installation and First Use Information
Java API	Installed as part of the LDAP client installation.
PL/SQL API	Installed as part of the Oracle9*i* Database Server. You must load it by using a script, called `catldap.sql`, located in `$ORACLE_HOME/rdbms/admin`.
C API	To build applications with the C API, you need to: Include the header file located at `$ORACLE_HOME/ldap/public/ldap.h`. Dynamically link to the library located at `$ORACLE_HOME/lib/libclntsh.so.9.0`.

User Management Functionality

This section describes user management functionality for the Java, PL/SQ, and C L LDAP APIs.

Directory-enabled applications need to access Oracle Internet Directory for the following user-related operations:

- User entry properties, which are stored as attributes of the user entry itself—in the same way, for example, as surname or home address
- Extended user preferences, which pertain to a user but are stored in a different location in the DIT. These properties can be further classified as:
  - Extended user properties common to all applications. These are stored in a common location in the Oracle Context.

  - Extended user properties specific to an application. These are stored in the application-specific **DIT**.
- Querying the group membership of a user
- Authenticating a user given a simple name and credential

A user is typically identified by the applications by one of the following techniques:

- A fully qualified LDAP **distinguished name (DN)**
- A **global unique identifier (GUID)**
- A simple user name along with the subscriber name

This section contains these topics:

User Management APIs

This section summarizes the user management functionality of each API.

Java API for User Management Functionality

As described in the example in the previous section, all user-related functionality is abstracted in a Java class called `oracle.ldap.util.User`. The high-level usage model for this functionality is:

1. Construct `oracle.ldap.util.User` object based on 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 itself.
4. Invoke `User.getExtendedProperties(DirContext, PropCategory, PropType)` to get the extended properties of the user. `PropCategory` here is either shared or application-specific. `PropType` is the object representing the type of property desired. If `PropType` is NULL, then 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 done by the application specific logic.

C API for User Management Functionality

Oracle Internet Directory 10*g* (9.0.4) does not support the C API for user management functionality.

User Authentication

This section describes user authentication functionality for the Java, PL/SQ, and C L LDAP APIs.

Java API for User Authentication

User authentication is a common LDAP operation that essentially compares a particular attribute and its attribute value. 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, in 10*g* (9.0.4), password policy applies only to the `userpassword` attribute.

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

```
        // 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
        }
```

C API for User Authentication

Oracle Internet Directory 10*g* (9.0.4) does not support the C API for user authentication functionality.

User Creation

This section describes user creation functionality for the Java, PL/SQL, and C L LDAP APIs.

Java API for User Creation

The `subscriber` class offers 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 defined above
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 10*g* (9.0.4) does not support the PL/SQL API for user creation functionality.

C API for User Creation

Oracle Internet Directory 10*g* (9.0.4) does not support the PL/SQL API for user creation functionality.

User Object Retrieval

This section describes user object retrieval functionality for the Java, PL/SQ, and C L 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 OID 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 users? 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 10*g* (9.0.4) does not support the PL/SQL API for user object retrieval functionality.

C API for User Object Retrieval

Oracle Internet Directory 10*g* (9.0.4) does not support the C API for user object retrieval functionality.

Group Management Functionality

This section describes the group management functionality for the Java, PL/SQ, and C L LDAP APIs.

Groups are modeled in Oracle Internet Directory as a collection of distinguished names. Directory-enabled applications need to access Oracle Internet Directory to get the properties of a group, and 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 the subscriber name

Identity Management Realm Functionality

This section describes the identity management realm functionality for the Java, PL/SQ, and C L 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 need to access Oracle Internet Directory to get realm properties—for example, 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

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 is contains a valid OID
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 allows 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 http://www.ietf.org/
- "A DNS RR for specifying the location of services (DNS SRV)", Internet RFC 2782 at http://www.ietf.org/

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 `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 case of DNS these inputs are:

- domain name
- discover method
- sslmode

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` object in above case). 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, then they are sorted on the basis of their weights and priorities. The SRV resource records contain host names and port numbers required for connection. This information is retrieved from the resource records 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 local machine on which the client is running. For example, if the client machine domain name is `mc1.acme.com`, then 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 is 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`, then, for non-SSL LDAP servers, the query 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:** "Oracle Directory Server Administration" in the *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 have been provided for overriding the default DSD behavior.

Table 3–2 Environment Variables for DSD Behavior

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+"'");
   }
  }
 }
}
```

Resource Information Management Functionality

To fulfill the requests of users, some Oracle components gather data from various repositories and services. To gather the data, these components require the following information:

- Information specifying the type of resource from which the data is to be gathered. The type of resource could be, for example, an Oracle Database. This is called resource type information.
- Information for connecting and authenticating users to the resources. This is called resource access information.

This section contains these topics:

Resource Type Information

Information about the resources that an application uses to service a user request is called resource type information. A resource type can be, for example, an Oracle9*i* Database Server or a Java Database Connectivity Pluggable Data Source. Resource type information includes such items as the class used to authenticate a user, the user identifier, and the password.

You specify resource type information by using the Oracle Internet Directory Self-Service Console.

Resource Access Information

Information for connecting and authenticating users to the databases is called resource access information. It is stored in an entry called a resource access descriptor (RAD) from which it can be retrieved and shared by various Oracle components.

For example, to service the request of a user for a sales report, Oracle Application Server Reports Services queries multiple databases. When it does this, it does the following:

1. Retrieves the necessary connect information from the RAD
2. Uses that information to connect to those databases and to authenticate the user requesting the data

Once it has done this, it compiles the report.

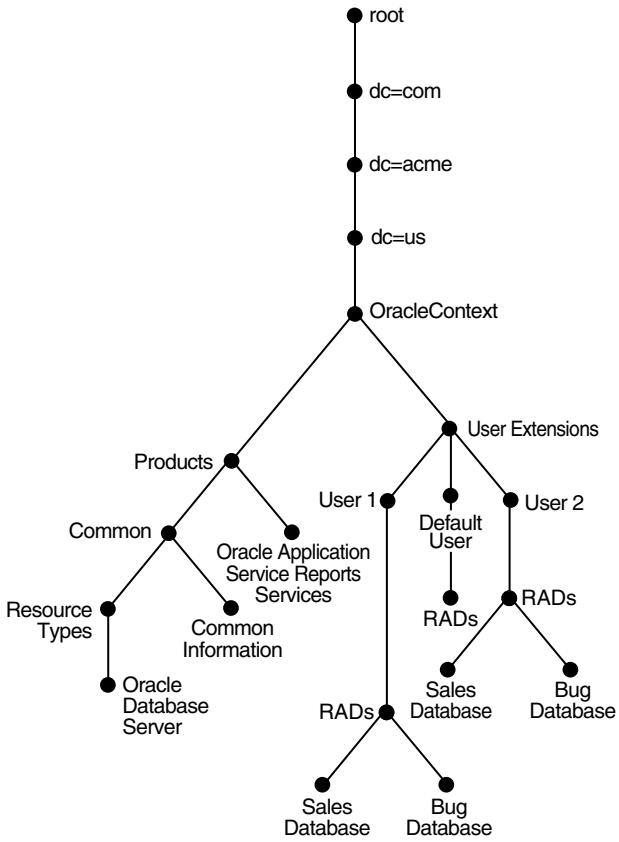
You specify resource access information by using the Oracle Internet Directory Self-Service Console. You can specify resource access information for each individual user or commonly for all users. In the latter case, all users connecting to a given application use, by default, the same information to connect to the necessary databases. Oracle Corporation recommends defining default resource access information whenever an application has its own integrated account

management—for example, where each user is defined withn the application itself by means of a unique songle sign-on user name.

Location of Resource Information in the DIT

Figure 3–4 shows where resource information is located in the DIT.

Figure 3–4 Placement of Resource Access and Resource Type Information in the DIT



As Figure 3–4 shows, the resource access and resource type information is stored in the Oracle Context.

Resource access information for each user is stored in the `cn=User Extensions` node in the Oracle Context. In this example, the `cn=User Extensions` node

contains resource access information for both the default user and for specific users. In the latter cases, the resource access information includes that needed for accessing both the Sales and the Bug databases.

Resource access information for each application is stored in the object identified by the application name—in this example, `cn=Oracle Application Server Reports Services, cn=Products,cn=Oracle Context,dc=us,dc=acme,dc=com`. This is the user information specific to that product.

Resource type information is stored in the container `cn=resource types, cn=common,cn=products,cn=Oracle Context`.

See Also:

- *Oracle Internet Directory Administrator's Guide*, for further information on how to set up and deploy RADs
- *Oracle Application Server Reports Services Publishing Reports to the Web*
- *Oracle Application Server Forms Services Deployment Guide*
- "Function get_user_extended_properties" on page 9-16
- *Oracle Internet Directory API Reference*for the following: `oracle.ldap.util.get_extended_properties`, `oracle.ldap.util.set_extended_properties`, and `oracle.ldap.util.create_extended_properties`

Richard's comment/question:

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 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 for RFCs 2829, 2831, and 2617

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

Context.SECURITY_AUTHENTICATION = "DIGEST-MD5".

Context.SECURITY_PRINCIPAL sets to the principal name.

The principal 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: "JNDI Sample Code" on page B-42

C APIs for SASL authentication using DIGEST-MD5

An LDAP client can use the provided C APIs to set up SASL digest MD5 to connect to the directory server.

See Also:

- "Authenticating to the Directory" on page 7-18
- "C API Usage for SASL-Based DIGEST-MD5 Authentication" on page 7-67

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 which 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.

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 the Oracle Directory Integration and Provisioning platform. 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 Internet Directory Administrator's 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 might 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 Oracle Internet 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 out the events that need to be sent or received.
- List out attributes of interest that an event should contain.
- Assign proper privileges to the application identity in the identity management realm so that the various events can be read from Oracle Internet Directory and propagated to it, and for applying change events to Oracle Internet Directory.
- Determine the interface name, interface type, and interface connection information. 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).

To clearly explain these general steps, we will consider a sample application.

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 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 4-6.

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 using 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 `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, then 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:

- INBOUND: from the application to Oracle Internet Directory
- OUTBOUND: from Oracle Internet Directory to the application, this is the default
- 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, and 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
then in an OID install it can be queried.
#                                  This value is stored in Root Oracle Context
in OID. This value is stored in
#                                  'orcldefaultsubscriber' attribute in
#                                  'dn: cn=Common,cn=Products,cn=OracleContext'
entry.)
#  %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 :  OID server hostname
#  LDAP_PORT :  OID server port number
#  BINDDN    :  cn=orcladmin
#  BINDPASSWD:  Password for orcladmin
#
#  After replacing the variables in the template this data can be loaded in OID
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  b'cos 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 below 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 below 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 below 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 below 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 below 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 below 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  entriy 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 earlier.

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 2 minutes, and a maximum of 100 events each time.

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 `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 PLSQL APIs to propagate or receive events.

Implementing the Interface Specification for the Employee Self Service Application

The interface is described in detail in Chapter 11, "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,telephonenu
mber)" \
event_subscription="IDENTITY:o=oracle,dc=com:DELETE " \
event_subscription="SUBSCRIPTION:cn=ESSA,cn=prducts,cn=oraclecontext,o=oracle,
dc=com:ADD(orclactivestartdate,orclactiveenddate,cn) \
event_
subscription="SUBSCRIPTION:cn=ESSA,cn=prducts,cn=oraclecontext,o=oracle,dc=com:M
ODIFY(orclactivestartdate,orclactiveenddate,cn) \
event_
subscription="SUBSCRIPTION:cn=ESSA,cn=prducts,cn=oraclecontext,o=oracle,dc=com:
DELETE"
```

Provisioning Integration Prerequisites

Applications used with Oracle Directory Provisioning Integration Service must be Oracle RDBMS-based and 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 as described in the chapter on the Oracle Directory Provisioning Integration Service in the *Oracle Internet Directory Administrator's Guide* in the Oracle Application Server documentation set.
- The Oracle Directory Integration and Provisioning platform
- Knowledge of SQL, PL/SQL, and database RPCs

In addition, Oracle Corporation recommends that you understand Oracle Application Server Single Sign-On concepts.

Development Usage Model for Provisioning Integration

This section gives an overview of the usage model for an agent for a provisioning-integrated application. It contains these topics:

Initiating Provisioning Integration

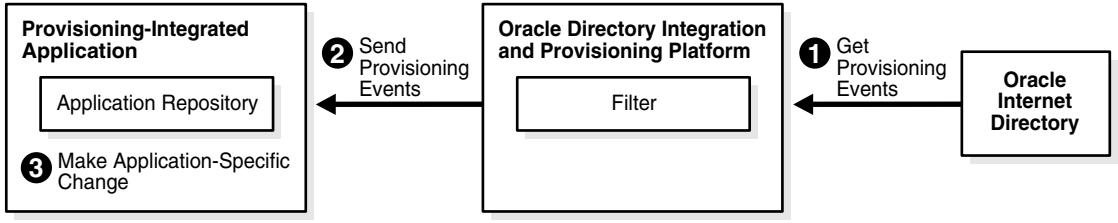
During application installation, the following information is provided to the Oracle Directory Provisioning Integration Service:

- Information to register the application entry in Oracle Internet Directory
- Information to register the application-specific database connect information with Oracle Internet Directory

Information for the Oracle Directory Provisioning Integration Service to service the application—for example, the kind of changes required, or scheduling properties. Figure 4–1 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 user and group information 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—based on the database connect information—by invoking a generic provisioning interface.
3. The generic provisioning interface invokes the 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 the 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 the 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

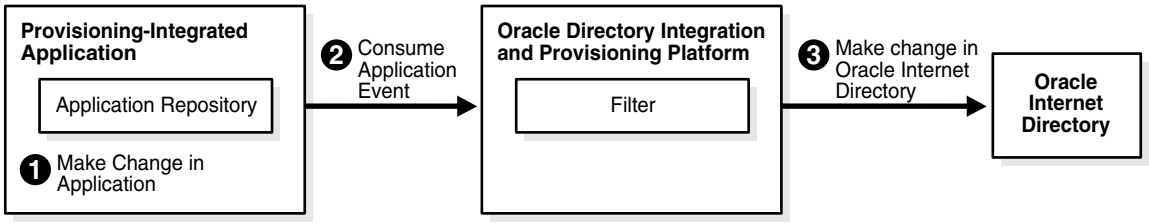


Figure 4–3 on page 4-19 shows the relationship between the services and the subscribed applications in a provisioning-integrated deployment.

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

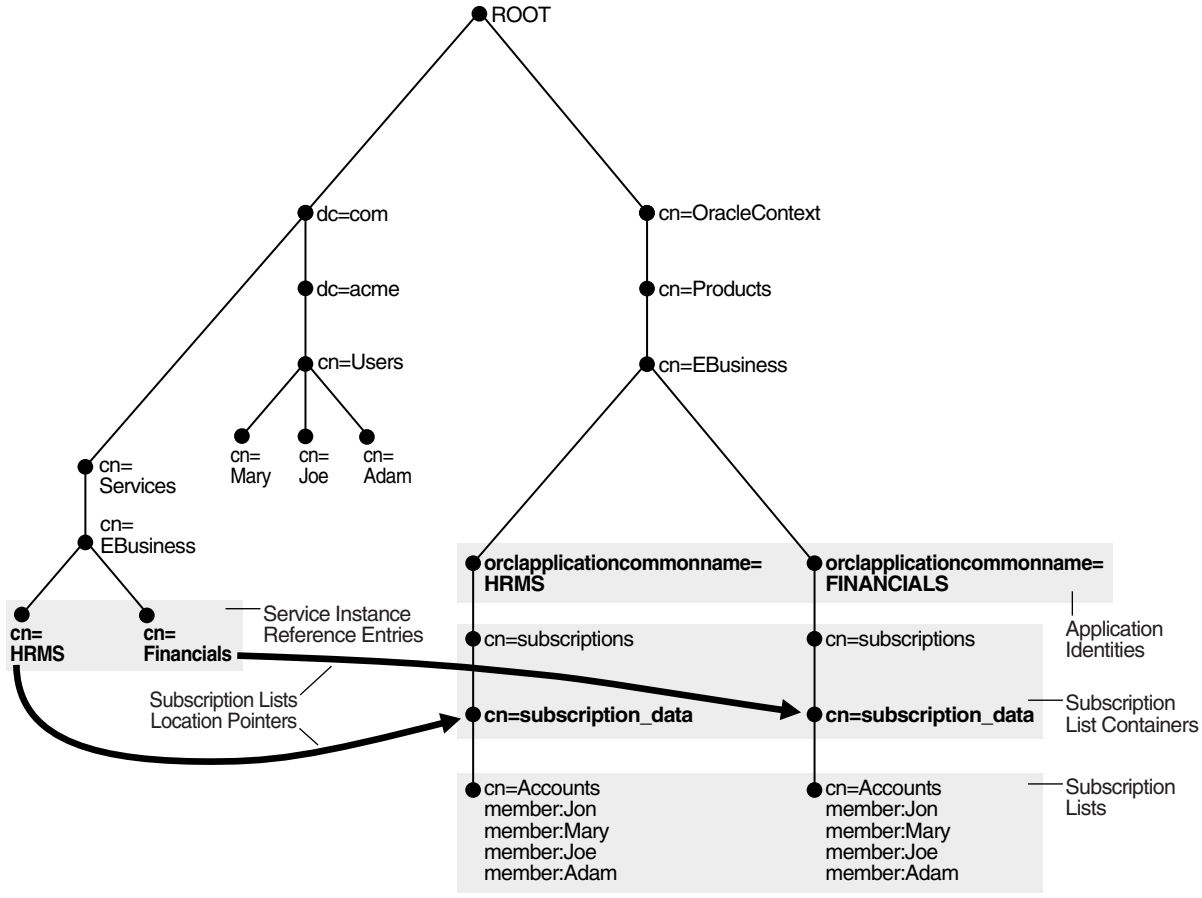


Figure 4–3 shows a DIT in which the entries for two services—Oracle Human Resources and Oracle Financials—point to their corresponding subscription list containers.

- 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 4-16 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.

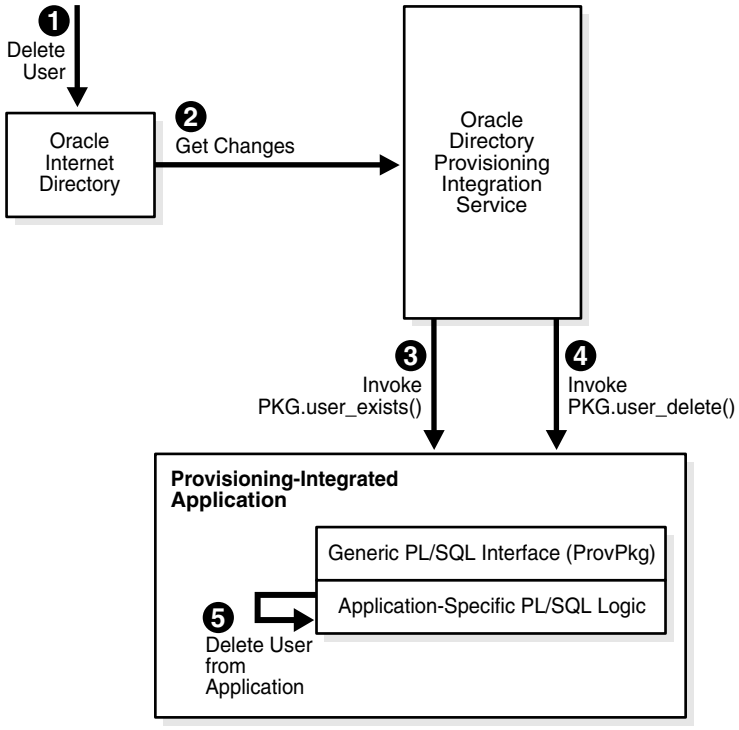
With the PL/SQL callback interface, then the application registers with the Oracle Directory Provisioning Integration Service and provides:

- 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 4–3 illustrates the system interactions for the PL/SQL callback interface.

Figure 4–4 PL/SQL Callback Interface



As Figure 4–3 shows, the deletion of 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, then 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.

   Step 5 is the responsibility of the provisioning-integrated application developer.

Extensible Event Definitions

This feature allows 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

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 (any) 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—that is, if an attribute has two values then two rows would be 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
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 Oracle Internet Directory Server Plug-ins

This chapter explains how to facilitate custom development by using the Oracle Internet Directory server plug-in framework.

This chapter contains these topics:

Introduction to Oracle Internet Directory Server Plug-ins

The plug-in framework for Oracle Internet Directory enables you to extend LDAP operations. For example:

- To authenticate a user when the user information is not stored in the directory server
- To attach certain custom operations to an LDAP operation. For example, some LDAP users may have different LDAP data value validation. For each `ldapadd` or `ldapmodify` operation, they may have different ways to validate the attribute values.

Prerequisite Knowledge for Developing Oracle Internet Directory Server Plug-ins

In order to develop Oracle Internet Directory plug-ins you should be familiar with:

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

Oracle Internet Directory Server Plug-ins Concepts

This section contains these topics:

About Directory Server Plug-ins

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, containing your own functions. (Currently, we support PL/SQL.)

You can write your own plug-in functions to extend the functionality of the Oracle Internet Directory server using the following methods:

- 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 example, you can 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, and calls your plug-in functions during the course of processing various LDAP requests.

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

About Server Plug-in Framework

Oracle Internet Directory server 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)

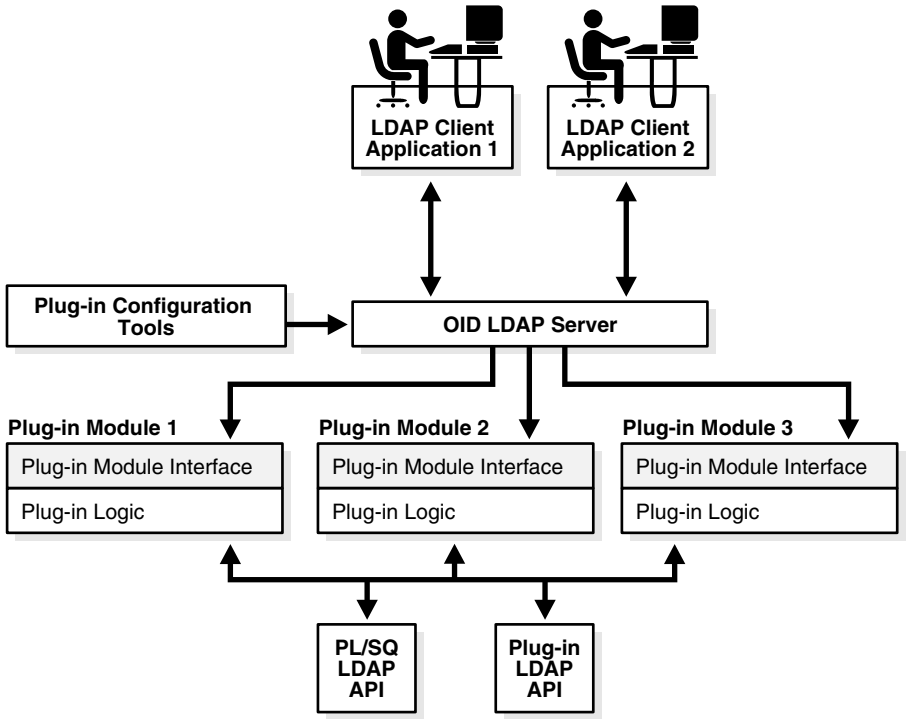
The steps to use the server plug-in framework are as follows:

1. Write a user-defined plug-in procedure. This plug-in module must be written in PL/SQL.

> **Note:** The PL/SQL language is currently supported.

2. Compile the plug-in module against the same database which 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.

Figure 5–1 Oracle Internet Directory Server Plug-in Framework?



Operation-Based Plug-ins Supported in Oracle Internet 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, then Oracle Internet Directory server does not rollback 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 will not be rolled back.

If the associated LDAP request fails, then the post plug-in will still be executed.

When-Operation Plug-ins

The OID server calls when-operation plug-in modules in addition to standard ldap processing. The main purpose of this type of plug-in is to augment existing functionality. Any extra operations that need to be thought of as part of an LDAP operation, that is, in the same LDAP transaction, must use the WHEN option. The when-operation plug-in is essentially in the same transaction as the associated LDAP request. If either the LDAP request or the plug-in program fails, then all the changes are rolled back.

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

- Add-on
- Replace

For example, for the `ldapcompare` operation, you can use the When Add-on type plug-in. Oracle Internet Directory server executes its server compare code and executes the plug-in module defined by the plug-in developer. For the Replace Type plug-in, Oracle Internet Directory does not execute its own compare code and relies on the plug-in module to do the comparison and pass back the compare result. The server comparison procedures are replaced by the plug-in module.

When Replace operation plug-ins are only supported in ldapadd, ldapcompare, ldapdelete, ldapmodify, and ldapbind. When add-on plug-ins are supported in ldapadd, ldapdelete, and ldapmodify.

Requirements for Oracle Internet Directory 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 with respect to other plug-ins or stored procedures in the same database schema. Plug-in names do not need to be unique with respect to other database schema objects, such as tables and views. For example, a database table and a plug-in can have the same name (however, to avoid confusion, this is not recommended).

Creating Plug-ins

The process to create a plug-in module is the same as to create a PL/SQL package. There is a plug-in specification part and a plug-in body part. Oracle Internet Directory defines the plug-in specification because the specification serves as the interface between Oracle Internet Directory server and custom plug-ins.

For security purposes and for the integrity of the LDAP server, plug-ins can only be compiled in ODS database schema against the database which serves as the backend database of the Oracle Internet Directory server.

Plug-in Module Interface Package Specifications

For different types of plug-ins, there are different package specifications defined. You can name the plug-in package. However, you must follow the signatures defined for each type of plug-in procedure.

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

See Also: Plug-in Module Interface Specifications on page 5-26 and Usage Model and Examples on page 5-20 for coding examples

The following table shows the parameters for different kinds of operation-based plug-ins.

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

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

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

See Also:

- Error Handling on page 5-15 for valid values for the return code and error message
- Directory Server Error Code Reference on page 5-30 for valid values for the OUT parameters return code
- Plug-in Module Interface Specifications 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 involves the following 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, then 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 attribute name list that controls if 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 ldap search filter type value need to be specified here. For example, if we specify `orclPluginEntryProperties:(&(objectclass=inetorgperson)(sn=Cezanne))`, then plug-in will not be 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

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

Attribute Name	Attribute Value	Mandatory?
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, then 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
orclPluginRequestNegGroup	A semicolon-separated group list that controls if the plug-in takes effect. You can use this group to specify who can NOT invoke the plug-in. For example, if you specify orclpluginrequestgroup: cn=security,cn=groups,dc=oracle,dc=com, when you register the plug-in, then the plug-in will not be 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

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

Attribute Name	Attribute Value	Mandatory?
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 in Oracle Internet Directory on page 5-4	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
```

Notes: The plug-in configuration entry, for example, `cn=plugin,cn=subconfigsubentry` metadata is not replicated in the replication environment to avoid creating inconsistent state.

Managing Plug-ins

This section explains modifying plug-ins and debugging plug-ins.

Modifying Plug-ins

Similar to 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, then 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 1/0.

Exception Handling

In each of the plug-in PL/SQL procedures, there must be an exception handling block to handle errors intelligently and, if possible, recover from them.

> **See Also:** PL/SQL Programming, Error Handling manual for information about how to use exceptions in a PL/SQL programming block

Error Handling

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

The valid values for the return code is as follows:

Error Code	Description
0	Success
Any number greater than zero (0)	Failure, **See Also** Directory Server Error Code Reference on page 5-30
-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, following the run,

Oracle Internet Directory examines the return code to determine if it must display the error message.

For example, if the value for the return code is 0, then the error message value is ignored. If the value of the return code is -1 or greater than zero, then the following `message` is either logged in the log file or displayed on the 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

When a plug-in exception is occurring, the following describes where the plug-in exception occurred and the Oracle Internet Directory server handling of the exception.

Table 5–4 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
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

When an LDAP operation fails, the following describes the ldap operation failure and the Oracle Internet Directory server handling of the failure.

Table 5–5 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.

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

LDAP Operation Fails in	Oracle Internet Directory Server Handling
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 Replacement	Changes made in the plug-in program body are rolled back.

Plug-in LDAP API

There are different methods for providing API access as follows:

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

In the Plug-in LDAP API, Oracle Internet Directory provides APIs for connecting back to the same Oracle Internet Directory server within the plug-in module. In other words, within the plug-in module, if you want to connect to any external directory server, you can use the DBMS_LDAP API. If you want to connect to the same Oracle Internet Directory server that is executing this plug-in itself, then you must use the Plug-in LDAP API for binding and authentication.

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

> **See Also:** Plug-in LDAP API Specifications on page 5-19 for coding examples

Plug-ins and Replication

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

- Plug-in metadata replicated to other nodes
- Use in the plug-in program of `ldapmodify`, `ldapadd`, or any other LDAP operation that changes the entries in the directory
- 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 you to supply the code that preserves tight security. For example, if you replace Oracle Internet 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

Oracle Internet Directory plug-in debugging will help you 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 the following command:

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

- To enable plug-in debugging, run the following command:

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

- After enabling plug-in debugging, you can use the command

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

  in the plug-in module code. The debugging message will be stored in the plug-in debugging table.

- To disable debugging, run the following command:

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

- To show debugging messages that you put in the plug-in module, run the following command:

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

- To delete all the debugging messages from the debugging table, run the following command:

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

Plug-in LDAP API Specifications

```
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;
```

Usage Model and Examples

This section contains two example situations about search query logging and synchronizing two directory information trees (DITs).

Example 1: Search Query Logging

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

Solution: Yes. Using the POST `ldapsearch` operational plug-in then the user can log all the `ldapsearch` commands. They can either log all the `ldapsearch` requests, or log all the `ldapsearch` requests if the search occurs under certain DNs (under a specific subtree).

To log all the `ldapsearch` commands, do the following:

1. Preparation.

   Log all of the `ldapsearch` results into a database table. This log table will have the following columns:

   - timestamp
   - baseDN
   - search scope
   - search filter
   - required attribute
   - search result

   Use the following 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 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 plug-in entry to Oracle Internet Directory server.

   Use the following to construct an LDIF file (`register_post_search.ldif`):

   ```
   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 dependent products under `cn=Products`, `cn=oraclecontext` where the users in these products have a one-to-one relationship in Oracle Internet Directory. If a user in the first DIT (product 1) is deleted, we want to delete the corresponding user in the other DIT (product 2) since a a relationship exists between these users.

Is there a way to set a trigger within Oracle Internet Directory that, on the event of deleting the user in the first DIT, will call or pass 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`, then the relationship between the two users in the different DITs is that they share the same ID attribute. Basically, inside of the post ldapdelete plug-in module, we use LDAP_PLUGIN and DBMS_LDAP APIs to delete the corresponding user in the 2nd 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.

1. Preparation.

   Assume the entries under both DITs have been added into 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
    (ldappluginсontext IN  ODS.plugincontext,
    result   IN  INTEGER,
    dn       IN  VARCHAR2,
    rc       OUT INTEGER,
    errormsg OUT VARCHAR2
    );
END LDAP_PLUGIN_EXAMPLE2;
/
```

3. Create plug-in package body.

```
CREATE OR REPLACE PACKAGE BODY LDAP_PLUGIN_EXAMPLE2 AS
PROCEDURE post_delete
    (ldappluginсontext 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 plug-in entry to Oracle Internet Directory server.

   Use the following to construct a 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 the following entry:

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

Database Type Definition and Plug-in Module Interface Specifications

This section gives examples of database object type definitions and LDAP_PLUGIN API Specifications.

This section contains these topics:

Database Object Type Definitions

This section contains the object definitions for those object types introduced in the Plug-in LDAP API. All these definitions are in Oracle Directory Server (ODS) database schema.

```
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 entryobj as object (
entryname varchar2(2000),
attr      attrlist
);
/

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 modobj as object (
operation integer,
type      varchar2(256),
vals      bvallist
);
/

create or replace type modlist as table of modobj;
/
```

Plug-in Module Interface Specifications

You must follow the procedure signature to use ldapbind, ldapsearch, ldapdelete, ldapadd, ldapcompare, and ldapmodify plug-ins.

```
CREATE or replace PACKAGE plugin_test1 AS

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 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 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 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
    );

END plugin_test1;
/
```

Directory Server Error Code Reference

```
-------------------------------------------------------------------------------
---Package specification for DBMS_LDAP
---     This is the primary interface used by various clients to
---     make LDAP requests
-------------------------------------------------------------------------
CREATE OR REPLACE PACKAGE DBMS_LDAP AS
-- …
     -- possible error codes we can return from LDAP server
     --
     SUCCESS                   CONSTANT NUMBER := 0;
     OPERATIONS_ERROR          CONSTANT NUMBER := 1;
     PROTOCOL_ERROR            CONSTANT NUMBER := 2;
     TIMELIMIT_EXCEEDED        CONSTANT NUMBER := 3;
     SIZELIMIT_EXCEEDED        CONSTANT NUMBER := 4;
     COMPARE_FALSE             CONSTANT NUMBER := 5;
     COMPARE_TRUE              CONSTANT NUMBER := 6;
     STRONG_AUTH_NOT_SUPPORTED CONSTANT NUMBER := 7;
     STRONG_AUTH_REQUIRED      CONSTANT NUMBER := 8;
     PARTIAL_RESULTS           CONSTANT NUMBER := 9;
     REFERRAL                  CONSTANT NUMBER := 10;
     ADMINLIMIT_EXCEEDED       CONSTANT NUMBER := 11;
     UNAVAILABLE_CRITIC        CONSTANT NUMBER := 12;
     NO_SUCH_ATTRIBUTE         CONSTANT NUMBER := 16;
     UNDEFINED_TYPE            CONSTANT NUMBER := 17;
     INAPPROPRIATE_MATCHING    CONSTANT NUMBER := 18;
     CONSTRAINT_VIOLATION      CONSTANT NUMBER := 19;
     TYPE_OR_VALUE_EXISTS      CONSTANT NUMBER := 20;
     INVALID_SYNTAX            CONSTANT NUMBER := 21;
     NO_SUCH_OBJECT            CONSTANT NUMBER := 32;
     ALIAS_PROBLEM             CONSTANT NUMBER := 33;
     INVALID_DN_SYNTAX         CONSTANT NUMBER := 34;
     IS_LEAF                   CONSTANT NUMBER := 35;
     ALIAS_DEREF_PROBLEM       CONSTANT NUMBER := 36;
     INAPPROPRIATE_AUTH        CONSTANT NUMBER := 48;
     INVALID_CREDENTIALS       CONSTANT NUMBER := 49;
     INSUFFICIENT_ACCESS       CONSTANT NUMBER := 50;
     BUSY                      CONSTANT NUMBER := 51;
     UNAVAILABLE               CONSTANT NUMBER := 52;
     UNWILLING_TO_PERFORM      CONSTANT NUMBER := 53;
     LOOP_DETECT               CONSTANT NUMBER := 54;
     NAMING_VIOLATION          CONSTANT NUMBER := 64;
     OBJECT_CLASS_VIOLATION    CONSTANT NUMBER := 65;
```

```
      NOT_ALLOWED_ON_NONLEAF    CONSTANT NUMBER := 66;
      NOT_ALLOWED_ON_RDN        CONSTANT NUMBER := 67;
      ALREADY_EXISTS            CONSTANT NUMBER := 68;
      NO_OBJECT_CLASS_MODS      CONSTANT NUMBER := 69;
      RESULTS_TOO_LARGE         CONSTANT NUMBER := 70;
      OTHER                     CONSTANT NUMBER := 80;
      SERVER_DOWN               CONSTANT NUMBER := 81;
      LOCAL_ERROR               CONSTANT NUMBER := 82;
      ENCODING_ERROR            CONSTANT NUMBER := 83;
      DECODING_ERROR            CONSTANT NUMBER := 84;
      TIMEOUT                   CONSTANT NUMBER := 85;
AUTH_UNKNOWN              CONSTANT NUMBER := 86;
FILTER_ERROR              CONSTANT NUMBER := 87;
USER_CANCELLED            CONSTANT NUMBER := 88;
PARAM_ERROR               CONSTANT NUMBER := 89;
NO_MEMORY                 CONSTANT NUMBER := 90;
```

6

Developing Applications Integrated with Oracle Delegated Administration Services

This chapter explains how developers can use the Oracle Delegated Administration Services URL service units to achieve integration with Oracle Delegated Administration Services.

It contains the following sections:

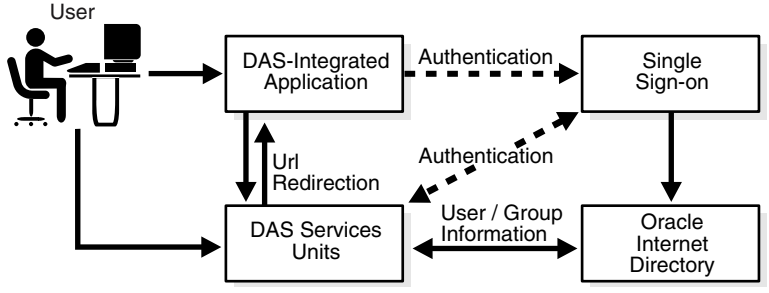
Introduction to the Delegated Administration Services

Oracle Delegated Administration Services are a set of pre-defined, Web-based service units for performing directory operations on behalf of a user. Oracle Delegated Administration Services units enable Oracle Internet Directory to use the self-service model for directory users to, for instance, update their own information in an employee directory.

Delegated Administration Services enable you to more easily develop tools for administering application data in the directory. They provide most of the functionality that directory-enabled applications require, such as creating a user entry, creating a group entry, searching for entries, and changing user passwords.

You can embed Delegated Administration Service units into your applications. For example, if you are building a Web portal, you can add Oracle Delegated Administration Services units to enable users to change application passwords stored in the directory. Each service unit has a corresponding URL stored in the directory. An application can invoke an Oracle Delegated Administration Services unit by URL discovery at runtime by querying the directory.

Figure 6–1 Overview of Delegated Administration Services



Benefits of Oracle Delegated Administration Services-Based Applications

There are three main areas where applications based on Oracle Delegated Administration Services are more advanced than those based on earlier types of APIs.

First, because Oracle Delegated Administration Services units are Web-based, an application developed with them are language-independent. In practice, 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). This tool reduces design and development time for these basic functions.

Third, Oracle Delegated Administration Services is integrated with Oracle Application Server Single Sign-On, so an application based on Oracle Delegated Administration Services is automatically authenticated with Oracle Application Server Single Sign-On. This means that an application using Oracle Delegated Administration Services can proxy as a user to query the directory on behalf of a user, for better security.

Developing Applications Integrated with Oracle Delegated Administration Services

This section contains these topics:

Prerequisites for Integration with Oracle Delegated Administration Services

For an application to integrate with Oracle Delegated Administration Services units, the following must be true:

- The application must be a Web-based GUI.
- The application must be integrated with Oracle Application Server Single Sign-On either through `mod_osso` or through partner application.
- The application has certain operations that need to be performed as the currently signed on user that can be leveraged from Oracle Delegated Administration Services.
- The application has users or groups stored in Oracle Internet Directory and can leverage Oracle Delegated Administration Services for user and group management.
- The application needs to be run under an Oracle Application Server infrastructure or middle-tier environment such that the discovery mechanism for the Oracle Delegated Administration Services URL is accessible.

Oracle Delegated Administration Services Integration Methodology and Considerations

Table 6–1discusses the various considerations for integrating an application with Oracle Delegated Administration Services.

Table 6–1 Condiserations 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 and identify integration points within the application GUI.
	Make necessary code changes to pass parameters to the Oracle Delegated Administration Services self-service units and also 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` Oracle Delegated Administration Services unit with a custom application. In the custom application page, `Create User` is shown as a link.

1. Identify the Oracle Delegated Administration Services URL base, by using the Java API string:

   ```
   baseUrl = Util.getDASUrl(ctx,DASURL_BASE).
   ```

   This API returns the Oracle Delegated Administration Services base URL in the following form: `http://host_name:port/`

2. Get the specific URL for the `Create User` Oracle Delegated Administration Services unit, by using the 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 our application.

   Next we will look at the parameters that can be customized for this unit. This unit takes following parameters:

Table 6–2 Oracle Delegated Administration Services URL Parameters

Parameter	Description
`homeURL`	The URL which is linked to the global button Home in the Oracle Delegated Administration Services unit. When the calling application specifies this value, you can click the Home button 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 case of Create User, once the user is created clicking on OK redirects the URL to this location. Hence the user navigation experience will be smooth.
`cancelURL`	This URL is linked with all the Cancel buttons shown in the 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/false. This will enable the section Assign Privileges in User or Group operation. If the enablePA is passed with value of true in the Create User page, then Assign Privileges to User section will also appear in 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 the following:

```
http://acme.mydomain.com:7777/oiddas/ui/oracle/ldap/das/admin/AppCreateUserI
nfoAdmin? 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

Oracle Delegated Administration Services List of Values (LOV) is implemented using JavaScript to invoke and pass values between the LOV calling window and Oracle Delegated Administration Services LOV page. The application invoking the LOV needs to open a popup window using JavaScript. Since the Java scripts have the security restrictions, data passing across the domains is not possible. Due to this limitation, only the pages in the same domain can access the Oracle Delegated Administration Services LOV units.

The base and the relative URL can be invoked the same way as Create User. Sample files are located at:

```
$ORACLE_HOME/ldap/das/samples/lov
```

This sample illustrates how the LOV can be invoked and data can be passed between the calling application and Oracle Delegated Administration Services unit. Complete illustration of the LOV invocation is beyond the scope of this chapter.

Java APIs Used to Access URLs

To discover the Oracle Delegated Administration Services URLs, Java APIs can be used. More details about the Java API are described in Chapter 3, "Developing Applications with Oracle Extensions to the Standard LDAP APIs" and Chapter 10,

"DAS_URL Interface Reference". The API functions which address the Oracle Delegated Administration Services URL discovery are:

- `getDASUrl`(DirContext ctx, String urlTypeDN)
- `getAllDASUrl`(DirContext ctx)

Part II

Oracle Internet Directory Programming Reference

Part II presents the Oracle-specific extensions to standard APIs. This part includes reference chapters that include classes, exceptions, and sample usage for the C, PL/SQL, Oracle Delegated Administration Services, and Provisioning Integration APIs. Further API reference material may be available on the product CD.

This part contains these chapters:

7

The C API for Oracle Internet Directory

This chapter introduces the Oracle Internet Directory C API and provides examples of how to use it.

It 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* (9.0.4) in the following modes:

- SSL—All communication securedby 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 7-65 for more details on how to use the two modes

This section contains these topics:

- Oracle Internet Directory SDK C API SSL Extensions
- Summary of LDAP C API

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 7–1 Arguments for SSL Interace 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 non zero return value indicates an error. The error code can be decoded by using the function ldap_err2string.

See Also: See "Sample C API Usage" on page 7-65

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* (9.0.4) of the API supports only the Oracle Wallet. You can create wallets by using Oracle Wallet Manager.

C API Reference

This section contains these topics:

Summary of LDAP C API

Table 7–2 DBMS_LDAP API Subprograms

Function or Procedure	Description
ber_free()	Free the memory allocated for a BerElement structure
ldap_abandon_ext ldap_abandon	Cancel an asynchronous operation

Table 7–2 (Cont.) DBMS_LDAP API Subprograms

Function or Procedure	Description
ldap_add_ext	Add a new entry to the directory
ldap_add_ext_s	
ldap_add	
ldap_add_s	
ldap_compare_ext	Compare entries in the directory
ldap_compare_ext_s	
ldap_compare	
ldap_compare_s	
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	Delete an entry from the directory
ldap_delete_ext_s	
ldap_delete	
ldap_delete_s	
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	Split up a distinguished name into its components
ldap_explode_rdn	
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

Table 7–2 (Cont.) DBMS_LDAP API Subprograms

Function or Procedure	Description
ora_ldap_get_cred_props	Retrieve properties associated with credential handle.
ldap_get_dn	Get the distinguished name for an entry
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_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.

Table 7–2 (Cont.) DBMS_LDAP API Subprograms

Function or Procedure	Description
ldap_result	Check the results of an asynchronous operation
ldap_msgfree	
ldap_msgtype	
ldap_msgid	
ldap_sasl_bind	General authentication to an LDAP server
ldap_sasl_bind_s	
ldap_search_ext	Search the directory
ldap_search_ext_s	
ldap_search	
ldap_search_s	
ldap_search_st	Search the directory with a timeout value
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.
ldap_simple_bind	Simple authentication to an LDAP server
ldap_simple_bind_s	
ldap_unbind_ext	End an LDAP session
ldap_unbind	
ldap_unbind_s	
ldap_value_free	Free the memory allocated for the string values of an attribute
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/
```

Functions

This section contains these topics:

Initializing an LDAP Session

ldap_init

ldap_open

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.

Syntax

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

Parameters

Table 7–3 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 which is separated from the host itself with a colon (:) character. The hosts will be tried in the order listed, stopping with the first one to which 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 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 then this parameter is ignored.

Usage Notes

ldap_init() and ldap_open() both return a "session handle," 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 in which case the

operating system error reporting mechanism can be checked 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 can be 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

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 7–4 lists and describes the paramters for LDAP session handle options.

Table 7–4 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 7–5. After the constant the actual hexadecimal value of the constant is listed in parentheses.
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.

Table 7–4 (Cont.) Parameters for LDAP Session Handle Options

Parameters	Description
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 7–5 lists and describes the constants for LDAP session handle options.

Table 7–5 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 or LDAP_OPT_OFF)	int *	This option determines whether referral cache is enabled or not. If this option is set to LDAP_OPT_ON then cache is enabled else 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 7–5 (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 or 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 reading the current setting using ldap_get_option(), a zero value means OFF and any nonzero value means ON. By default, this option is ON.

Table 7–5 (Cont.) Constants

Constant	Type for invalue parameter	Type for outvalue parameter	Description
LDAP_OPT_ RESTART (0x09)	void * (LDAP_ OPT_ON or LDAP_OPT_OFF)	int *	Determines whether LDAP I/O operations are automatically restarted if they stop prematurely. 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 reading the current setting using ldap_get_option(), a zero value means OFF and any nonzero value means ON. This option is useful if an LDAP I/O operation can be interrupted prematurely, for example by a timer going off, or other interrupt. By default, this option is OFF.
LDAP_OPT_ PROTOCOL_ VERSION (0x11)	int *	int *	This option indicates the version of the LDAP protocol used when communicating with the primary LDAP server. It SHOULD be one of the constants 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 7-16
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 7-16
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 and 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 host name parameter to ldap_init() for the allowed syntax.

Table 7–5 (Cont.) Constants

Constant	Type for invalue parameter	Type for outvalue parameter	Description
LDAP_OPT_ ERROR_NUMBER (0x31)	int *	int *	The code of the most recent LDAP error that occurred for this session.
LDAP_OPT_ ERROR_STRING (0x32)	char *	char **	The message returned with the most recent LDAP error that occurred for this session.
LDAP_OPT_ MATCHED_DN (0x33)	char *	char **	The matched DN value returned with the most recent LDAP error that occurred for 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 implementors:

```
#define LDAP_OPT_PRIVATE_EXTENSION_BASE 0x4000  /* to 0x7FFF inclusive */
```

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 7–6.

Table 7–6 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 and/or client. Note that the LDAP unbind and abandon operations have no server response, so 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_set_option" on page 7-10. 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 7-10, 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 4-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 4-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.

Authenticating to the Directory

The following functions are used to authenticate an LDAP client to an LDAP directory server.

ldap_sasl_bind

ldap_sasl_bind_s

ldap_simple_bind

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 7–7 lists and describes the parameters for authenticating to the directory.

Table 7–7 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

Table 7–7 (Cont.) Parameters for Authenticating to the Directory

Parameter	Description
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(), described in "ldap_result" on page 7-47, 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" on page 7-50 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.

This function among other arguments accepts

- 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 requies 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.

The following functions are used to create and manage SASL credential handles.

ora_ldap_create_cred_hdl

ora_ldap_set_cred_props

ora_ldap_get_cred_props

ora_ldap_free_cred_hdl

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()` 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 `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                  propType,
        void             *   inProperty
);
OraLdapHandle ora_ldap_get_cred_props
(
        OraLdapClientCtx *   clientCtx,
        OraLdapHandle   cred,
        int                     propType,
        void                  *   outProperty
);

OraLdapHandle ora_ldap_free_cred_hdl
(
```

```
        OraLdapClientCtx *   clientCtx,
        OraLdapHandle   cred
);
```

Parameters

Table 7–8 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.
credType	Type of credential handle specific to SASL mechanism.
cred	Credential handle containing SASL credentials needed for a specific SASL mechanism for SASL authentication.
propType	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

The following function can be used for SASL authentication:

ora_ldap_init_SASL

This function performs SASL 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 7–9 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 who needs to be authenticated.
mechanism	SASL mechanism.
cred	Credentials needed for SASL authentication.
serverctrls	List of LDAP server controls
clientctrls	List of client controls

Closing the Session

The following functions are used to unbind from the directory, close open connections, and dispose of the session handle.

ldap_unbind_ext

ldap_unbind

ldap_unbind_s

Syntax

```
int ldap_unbind_ext( LDAP *ld, LDAPControl **serverctrls,
LDAPControl **clientctrls );
int ldap_unbind( LDAP *ld );
int ldap_unbind_s( LDAP *ld );
```

Parameters

Table 7–10 Parameters for Closing the Session

Parameter	Description
ld	The session handle
serverctrls	List of LDAP server controls
clientctrls	List of client controls

Usage Notes

The 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.

Performing LDAP Operations

These functions are used to search the LDAP directory, returning a requested set of attributes for each entry matched:

ldap_search_ext

ldap_search_ext_s

ldap_search

ldap_search_s

ldap_search_st

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 7–11 lists and describes the parameters for search operations.

Table 7–11 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.
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.

Table 7–11 (Cont.) Parameters for Search Operations

Parameter	Description
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 7–11 (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.

Usage Notes

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(), described in "ldap_result" on page 7-47, 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(), described in "ldap_result" on page 7-47, 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(), 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" on page 7-50 for more information about possible errors and how to interpret them.

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. attrs 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. attrs contains the list of attributes to return for each child entry.

ldap_compare_ext

ldap_compare_ext_s

ldap_compare

ldap_compare_s

These routines are used to compare a given attribute value assertion against an LDAP entry.

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 7–12 lists and describes the parameters for compare operations.

Table 7–12 Parameters for Compare Operations

Parameter	Description
ld	The session handle.

Table 7–12 (Cont.) Parameters for Compare Operations

Parameter	Description
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.
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.

Usage Notes

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(), described in "ldap_result" on page 7-47, 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(), described in "ldap_result" on page 7-47, 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" on page 7-50 for more information about possible errors and how to interpret them.

ldap_modify_ext

ldap_modify_ext_s

ldap_modify

ldap_modify_s

These routines are used to modify an existing LDAP entry.

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 7–13 lists and describes the parameters for modify operations.

Table 7–13 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 7–14 lists and describes the fields in the LDAPMod structure.

Table 7–14 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.

Table 7–14 (Cont.) Fields in LDAPMod Structure

Field	Description
mod_type	The type of the attribute to modify.
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.

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 `*msgid`p. A subsequent call to `ldap_result()`, described in "ldap_result" on page 7-47, 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()`, described in "ldap_result" on page 7-47, 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" on page 7-50 for more information about possible errors and how to interpret them.

ldap_rename

ldap_rename_s

These routines are used to change the name of an entry.

```
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 7–15 lists and describes the parameters for rename operations.

Table 7–15 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.
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(s) is to be removed, if zero indicating that the old RDN value(s) 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.

Usage Notes

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()`, described in "ldap_result" on page 7-47, 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" on page 7-50 for more information about possible errors and how to interpret them.

ldap_add_ext

ldap_add_ext_s

ldap_add

ldap_add_s

These functions are used to add entries to the LDAP directory.

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 7–16 lists and describes the parameters for add operations.

Table 7–16 Parameters for Add Operations

Parameter	Description
ld	The session handle.
dn	The name of the entry to add.
attrs	The entry's 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 ORed 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.

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()`, described in "ldap_result" on page 7-47, 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(), described in "ldap_result" on page 7-47, 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" on page 7-50 for more information about possible errors and how to interpret them.

ldap_delete_ext

ldap_delete_ext_s

ldap_delete

ldap_delete_s

These functions are used to delete a leaf entry from the LDAP directory.

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 7–17 lists and describes the parameters for delete operations.

Table 7–17 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.

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()`, described in "ldap_result" on page 7-47, 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()`, described in "ldap_result" on page 7-47, 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" on page 7-50 for more information about possible errors and how to interpret them.

ldap_extended_operation

ldap_extended_operation_s

These routines allow extended LDAP operations to be passed to the server, providing a general protocol extensibility mechanism.

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 7–18 lists and describes the parameters for extended operations.

Table 7–18 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)
serverctrls	List of LDAP server controls
clientctrls	List of client controls

Table 7–18 (Cont.) Parameters for Extended Operations

Parameter	Description
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.

Usage Notes

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()`, described in "ldap_result" on page 7-47, 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" on page 7-50 for more information about possible errors and how to interpret them.

Abandoning an Operation

ldap_abandon_ext

ldap_abandon

These calls are used to abandon an operation in progress:

Syntax

```
int ldap_abandon_ext
(
    LDAP            *ld,
    int             msgid,
    LDAPControl     **serverctrls,
    LDAPControl     **clientctrls
);

int ldap_abandon
(
    LDAP            *ld,
    int             msgid
);
```

Parameters

Table 7–19 lists and describes the parameters for abandoning an operation.

Table 7–19 Parameters for Abandoning an Operation

Parameter	Description
ld	The session handle.
msgid	The message id of the request to be abandoned.
serverctrls	List of LDAP server controls.
clientctrls	List of client controls.

Usage Notes

`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.

> **See Also:** "Handling Errors and Parsing Results" on page 7-50 for more information about possible errors and how to interpret them.

Obtaining Results and Peeking Inside LDAP Messages

ldap_result

ldap_msgfree

ldap_msgtype

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 7–20 lists and describes the parameters for obtaining results and peeling inside LDAP messages.

Table 7–20 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(s) 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

ldap_parse_result

ldap_parse_sasl_bind_result

ldap_parse_extended_result

ldap_err2string

These calls are used to extract information from results and handle errors returned by other LDAP API routines. 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.

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 use of the following routines is deprecated and more complete descriptions can be found in RFC 1823:

```
int ldap_result2error
(
    LDAP            *ld,
    LDAPMessage     *res,
    int             freeit
);
void ldap_perror( LDAP *ld, const char *msg );
```

Parameters

Table 7–21 lists and describes parameters for handling errors and parsing results.

Table 7–21 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.

Table 7–21 (Cont.) Parameters for Handling Errors and Parsing Results

Parameter	Description
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.
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

Additional parameters for the deprecated routines are not described. Interested readers are referred to RFC 1823.

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.

Stepping Through a List of Results

These routines are used to step through the list of messages in a result chain returned by ldap_result().

ldap_first_message

ldap_next_message

For search operations, the result chain can actually 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 7–22 lists and describes the parameters for stepping through a list of results.

Table 7–22 Parameters for Stepping Through a List of Results

Parameter	Description
ld	The session handle.

Table 7–22 (Cont.) Parameters for Stepping Through a List of Results

Parameter	Description
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

The following calls are used to parse the entries and references returned by `ldap_search()` and friends. 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

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 7–23 lists and describes the parameters or retrieving entries and continuation references from a search result chain, and for counting entries returned.

Table 7–23 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().

Table 7–23 (Cont.) Parameters for Retrieving Entries and Continuation References from a Search Result Chain, and for Counting Entries Returned

Parameter	Description
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

ldap_next_attribute

These calls are used to step through the list of attribute types returned with an entry.

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 7–24 lists and describes the parameters for stepping through attribute types returned with an entry.

Table 7–24 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.

Table 7–24 (Cont.) Parameters for Stepping Through Attribute Types Returned with

Parameter	Description
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()` will return `NULL` when the end of the attributes is reached, or if there is an error, in which case the error parameters in the session handle ld will be 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

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 7–25 lists and describes the parameters for retrieving and counting attribute values.

Table 7–25 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().
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

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 7–26 lists and describes the parameters for retrieving, exploding, and converting entry names.

Table 7–26 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()` will return `NULL` if there is some error parsing the DN, setting 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 7–27 lists and describes the parameters for extracting LDAP control from an entry.

Table 7–27 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()`.

Table 7–27 (Cont.) Parameters for Extracting LDAP Controls from an Entry

Parameters	Description
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

`ldap_parse_reference()` is used 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 7–28 lists and describes parameters for extracting referrals and controls from a `searchresultreference` message.

Table 7–28 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 SHOULD be 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

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <netdb.h>
#include <gsle.h>
#include <gslc.h>
#include <gsld.h>
#include "gslcc.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 <string.h>
#include <ctype.h>
#include <netdb.h>
#include <gsle.h>
#include <gslc.h>
#include <gsld.h>
#include "gslcc.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/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 OID 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 */
```

Building Applications with the C API

This section contains these topics:

- Required Header Files and Libraries
- Building a Sample Search Tool

Required Header Files and Libraries

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.9.0`.

Building a Sample Search Tool

The Oracle Internet Directory SDK 10*g* (9.0.4) provides a sample command line tool, samplesearch, for demonstrating how to use the C API to build applications. You can use samplesearch to perform LDAP searches in either SSL or non-SSL mode.

You can find the source file (`samplesearch.c`) and the make file (`demo_ldap.mk`) in the following directory: *$ORACLE_HOME*`/ldap/demo`.

To build the sample search tool, enter the following command:

```
make -f demo_ldap.mk build EXE=samplesearch OBJS=samplesearch.o
```

Note: You can use this make file to build other client applications by using the C API. Replace `samplesearch` with the name of the binary you want to build, and `samplesearch.o` with your own object file.

The sample code for samplesearch is:

```
/*
   NAME
     s0gsldsearch.c - <one-line expansion of the name>
   DESCRIPTION
     <short description of component this file declares/defines>
   PUBLIC FUNCTION(S)
     <list of external functions declared/defined - with one-line descriptions>
   PRIVATE FUNCTION(S)
     <list of static functions defined in .c file - with one-line descriptions>
   RETURNS
     <function return values, for .c file with single function>
   NOTES
     <other useful comments, qualifications, and so on>
*/
#include <stdio.h>
```

```
#include <string.h>
#include <ctype.h>
#include <netdb.h>
#include "ldap.h"

#define DEFSEP"="
#define LDAPSEARCH_BINDDN       NULL
#define LDAPSEARCH_BASE         DEFAULT_BASE
#define DEFAULT_BASE    "o=oracle, c=US"

#ifdef LDAP_DEBUG
extern int ldap_debug, lber_debug;
#endif /* LDAP_DEBUG */

usage( s )
char*s;
{
    fprintf( stderr, "usage: %s [options] filter [attributes...]\nwhere:\n", s
);
    fprintf( stderr, "    filter\tRFC-1558 compliant LDAP search filter\n" );
    fprintf( stderr, "    attributes\twhitespace-separated list of attributes to
retrieve\n" );
    fprintf( stderr, "\t\t(if no attribute list is given, all are retrieved)\n"
);
    fprintf( stderr, "options:\n" );
    fprintf( stderr, "    -n\t\tshow what would be done but don't actually
search\n" );
    fprintf( stderr, "    -v\t\trun in verbose mode (diagnostics to standard
output)\n" );
    fprintf( stderr, "    -t\t\twrite values to files in /tmp\n" );
    fprintf( stderr, "    -u\t\tinclude User Friendly entry names in the
output\n" );
    fprintf( stderr, "    -A\t\tretrieve attribute names only (no values)\n" );
    fprintf( stderr, "    -B\t\tdo not suppress printing of non-ASCII values\n"
);
    fprintf( stderr, "    -L\t\tprint entries in LDIF format (-B is implied)\n"
);
#ifdef LDAP_REFERRALS
    fprintf( stderr, "    -R\t\tdo not automatically follow referrals\n" );
#endif /* LDAP_REFERRALS */
    fprintf( stderr, "    -d level\tset LDAP debugging level to `level'\n" );
    fprintf( stderr, "    -F sep\tprint `sep' instead of `=' between attribute
names and values\n" );
    fprintf( stderr, "    -S attr\tsort the results by attribute `attr'\n" );
    fprintf( stderr, "    -f file\tperform sequence of searches listed in
```

```
`file'\n" );
    fprintf( stderr, "    -b basedn\tbase dn for search\n" );
    fprintf( stderr, "    -s scope\tone of base, one, or sub (search scope)\n"
);
    fprintf( stderr, "    -a deref\tone of never, always, search, or find (alias
dereferencing)\n" );
    fprintf( stderr, "    -l time lim\ttime limit (in seconds) for search\n" );
    fprintf( stderr, "    -z size lim\tsize limit (in entries) for search\n" );
    fprintf( stderr, "    -D binddn\tbind dn\n" );
    fprintf( stderr, "    -w passwd\tbind passwd (for simple authentication)\n"
);
#ifdef KERBEROS
    fprintf( stderr, "    -k\t\tuse Kerberos instead of Simple Password
authentication\n" );
#endif
    fprintf( stderr, "    -h host\tldap server\n" );
    fprintf( stderr, "    -p port\tport on ldap server\n" );
    fprintf( stderr, "    -W Wallet\tWallet location\n" );
    fprintf( stderr, "    -P Wpasswd\tWallet Password\n" );
    fprintf( stderr, "    -U SSLAuth\tSSL Authentication Mode\n" );
    return;
}

static char*binddn = LDAPSEARCH_BINDDN;
static char*passwd = NULL;
static char*base = LDAPSEARCH_BASE;
static char*ldaphost = NULL;
static intldapport = LDAP_PORT;
static char*sep = DEFSEP;
static char*sortattr = NULL;
static intskipsortattr = 0;
static intverbose, not, includeufn, allow_binary, vals2tmp, ldif;
/* TEMP */

main( argc, argv )
intargc;
char**argv;
{
    char*infile, *filtpattern, **attrs, line[ BUFSIZ ];
    FILE*fp;
    intrc, i, first, scope, kerberos, deref, attrsonly;
    intldap_options, timelimit, sizelimit, authmethod;
    LDAP*ld;
    extern char*optarg;
    extern intoptind;
```

```
    charlocalHostName[MAXHOSTNAMELEN + 1];
    char *sslwrl = NULL;
    char*sslpasswd = NULL;
int sslauth=0,err=0;

    infile = NULL;
    deref = verbose = allow_binary = not = kerberos = vals2tmp =
    attrsonly = ldif = 0;
#ifdef LDAP_REFERRALS
    ldap_options = LDAP_OPT_REFERRALS;
#else /* LDAP_REFERRALS */
    ldap_options = 0;
#endif /* LDAP_REFERRALS */
    sizelimit = timelimit = 0;
    scope = LDAP_SCOPE_SUBTREE;

    while (( i = getopt( argc, argv,
#ifdef KERBEROS
    "KknuvtRABLD:s:f:h:b:d:p:F:a:w:l:z:S:"
#else
    "nuvtRABLD:s:f:h:b:d:p:F:a:w:l:z:S:W:P:U:"
#endif
    )) != EOF ) {
switch( i ) {
case 'n':/* do Not do any searches */
    ++not;
    break;
case 'v':/* verbose mode */
    ++verbose;
    break;
case 'd':
#ifdef LDAP_DEBUG
    ldap_debug = lber_debug = atoi( optarg );/* */
#else /* LDAP_DEBUG */
    fprintf( stderr, "compile with -DLDAP_DEBUG for debugging\n" );
#endif /* LDAP_DEBUG */
    break;
#ifdef KERBEROS
case 'k':/* use kerberos bind */
    kerberos = 2;
    break;
case 'K':/* use kerberos bind, 1st part only */
    kerberos = 1;
    break;
#endif
```

```
case 'u':/* include UFN */
    ++includeufn;
    break;
case 't':/* write attribute values to /tmp files */
    ++vals2tmp;
    break;
case 'R':/* don't automatically chase referrals */
#ifdef LDAP_REFERRALS
    ldap_options &= ~LDAP_OPT_REFERRALS;
#else /* LDAP_REFERRALS */
    fprintf( stderr,
    "compile with -DLDAP_REFERRALS for referral support\n" );
#endif /* LDAP_REFERRALS */
    break;
case 'A':/* retrieve attribute names only -- no values */
    ++attrsonly;
    break;
case 'L':/* print entries in LDIF format */
    ++ldif;
    /* fall through -- always allow binary when outputting LDIF */
case 'B':/* allow binary values to be printed */
    ++allow_binary;
    break;
case 's':/* search scope */
    if ( strncasecmp( optarg, "base", 4 ) == 0 ) {
scope = LDAP_SCOPE_BASE;
    } else if ( strncasecmp( optarg, "one", 3 ) == 0 ) {
scope = LDAP_SCOPE_ONELEVEL;
    } else if ( strncasecmp( optarg, "sub", 3 ) == 0 ) {
scope = LDAP_SCOPE_SUBTREE;
    } else {
fprintf( stderr, "scope should be base, one, or sub\n" );
usage( argv[ 0 ] );
                exit(1);
    }
    break;

case 'a':/* set alias deref option */
    if ( strncasecmp( optarg, "never", 5 ) == 0 ) {
deref = LDAP_DEREF_NEVER;
    } else if ( strncasecmp( optarg, "search", 5 ) == 0 ) {
deref = LDAP_DEREF_SEARCHING;
    } else if ( strncasecmp( optarg, "find", 4 ) == 0 ) {
deref = LDAP_DEREF_FINDING;
    } else if ( strncasecmp( optarg, "always", 6 ) == 0 ) {
```

```
deref = LDAP_DEREF_ALWAYS;
    } else {
fprintf( stderr, "alias deref should be never, search, find, or always\n" );
usage( argv[ 0 ] );
                exit(1);
    }
    break;

case 'F':/* field separator */
    sep = (char *)strdup( optarg );
    break;
case 'f':/* input file */
    infile = (char *)strdup( optarg );
    break;
case 'h':/* ldap host */
    ldaphost = (char *)strdup( optarg );
    break;
case 'b':/* searchbase */
    base = (char *)strdup( optarg );
    break;
case 'D':/* bind DN */
    binddn = (char *)strdup( optarg );
    break;
case 'p':/* ldap port */
    ldapport = atoi( optarg );
    break;
case 'w':/* bind password */
    passwd = (char *)strdup( optarg );
    break;
case 'l':/* time limit */
    timelimit = atoi( optarg );
    break;
case 'z':/* size limit */
    sizelimit = atoi( optarg );
    break;
case 'S':/* sort attribute */
    sortattr = (char *)strdup( optarg );
    break;
case 'W':/* Wallet URL */
    sslwrl = (char *)strdup( optarg );
    break;
case 'P':/* Wallet password */
    sslpasswd = (char *)strdup( optarg );
    break;
case 'U':/* SSL Authentication Mode */
```

```
    sslauth = atoi( optarg );
    break;
default:
    usage( argv[0] );
            exit(1);
            break;
}
    }

    if ( argc - optind < 1 ) {
usage( argv[ 0 ] );
        exit(1);
    }
    filtpattern = (char *)strdup( argv[ optind ] );
    if ( argv[ optind + 1 ] == NULL ) {
attrs = NULL;
    } else if ( sortattr == NULL || *sortattr == '\0' ) {
        attrs = &argv[ optind + 1 ];
    } else {
for ( i = optind + 1; i < argc; i++ ) {
    if ( strcasecmp( argv[ i ], sortattr ) == 0 ) {
break;
    }
}
if ( i == argc ) {
skipsortattr = 1;
argv[ optind ] = sortattr;
} else {
optind++;
}
        attrs = &argv[ optind ];
    }

    if ( infile != NULL ) {
if ( infile[0] == '-' && infile[1] == '\0' ) {
    fp = stdin;
} else if (( fp = fopen( infile, "r" )) == NULL ) {
    perror( infile );
    exit( 1 );
}
    }

    if (ldaphost == NULL) {
        if (gethostname(localHostName, MAXHOSTNAMELEN) != 0) {
                perror("gethostname");
```

```
                exit(1);
            }
            ldaphost = localHostName;
        }

        if ( verbose ) {
    printf( "ldap_open( %s, %d )\n", ldaphost, ldapport );
        }

        if (( ld = ldap_open( ldaphost, ldapport )) == NULL ) {
    perror( ldaphost );
    exit( 1 );
        }

        if (sslauth > 1)
        {
           if (!sslwrl || !sslpasswd)
       {
              printf ("Null Wallet or password given\n");
              exit (0);
           }
        }
        if (sslauth > 0)
        {
            if (sslauth == 1)
                sslauth = GSLC_SSL_NO_AUTH;
            else if (sslauth == 2)
                sslauth = GSLC_SSL_ONEWAY_AUTH;
            else if (sslauth == 3)
                sslauth = GSLC_SSL_TWOWAY_AUTH;
            else
            {
        printf(" Wrong SSL Authenication Mode Value\n");
        exit(0);
            }

        err = ldap_init_SSL(&ld->ld_sb,sslwrl,sslpasswd,sslauth);
        if(err != 0)
    {
        printf(" %s\n", ldap_err2string(err));
        exit(0);
    }
        }

        ld->ld_deref = deref;
```

```
    ld->ld_timelimit = timelimit;
    ld->ld_sizelimit = sizelimit;
    ld->ld_options = ldap_options;

    if ( !kerberos ) {
authmethod = LDAP_AUTH_SIMPLE;
    } else if ( kerberos == 1 ) {
authmethod = LDAP_AUTH_KRBV41;
    } else {
authmethod =  LDAP_AUTH_KRBV4;
    }
    if ( ldap_bind_s( ld, binddn, passwd, authmethod ) != LDAP_SUCCESS ) {
ldap_perror( ld, "ldap_bind" );
exit( 1 );
    }

    if ( verbose ) {
printf( "filter pattern: %s\nreturning: ", filtpattern );
if ( attrs == NULL ) {
    printf( "ALL" );
} else {
    for ( i = 0; attrs[ i ] != NULL; ++i ) {
printf( "%s ", attrs[ i ] );
    }
}
putchar( '\n' );
    }

    if ( infile == NULL ) {
rc = dosearch( ld, base, scope, attrs, attrsonly, filtpattern, "" );
    } else {
rc = 0;
first = 1;
while ( rc == 0 && fgets( line, sizeof( line ), fp ) != NULL ) {
    line[ strlen( line ) - 1 ] = '\0';
    if ( !first ) {
putchar( '\n' );
    } else {
first = 0;
    }
    rc = dosearch( ld, base, scope, attrs, attrsonly, filtpattern,
    line );
}
if ( fp != stdin ) {
    fclose( fp );
```

```
}
    }

    ldap_unbind( ld );
    exit( rc );
}

dosearch( ld, base, scope, attrs, attrsonly, filtpatt, value )
    LDAP*ld;
    char*base;
    intscope;
    char**attrs;
    intattrsonly;
    char*filtpatt;
    char*value;
{
    charfilter[ BUFSIZ ], **val;
    intrc, first, matches;
    LDAPMessage*res, *e;

    sprintf( filter, filtpatt, value );

    if ( verbose ) {
printf( "filter is: (%s)\n", filter );
    }

    if ( not ) {
return( LDAP_SUCCESS );
    }

    if ( ldap_search( ld, base, scope, filter, attrs, attrsonly ) == -1 ) {
ldap_perror( ld, "ldap_search" );
return( ld->ld_errno );
    }

    matches = 0;
    first = 1;
    while ( (rc = ldap_result( ld, LDAP_RES_ANY, sortattr ? 1 : 0, NULL, &res ))
    == LDAP_RES_SEARCH_ENTRY ) {
matches++;
e = ldap_first_entry( ld, res );
if ( !first ) {
    putchar( '\n' );
} else {
    first = 0;
```

```
}
print_entry( ld, e, attrsonly );
ldap_msgfree( res );
    }
    if ( rc == -1 ) {
ldap_perror( ld, "ldap_result" );
return( rc );
    }
    if (( rc = ldap_result2error( ld, res, 0 )) != LDAP_SUCCESS ) {
        ldap_perror( ld, "ldap_search" );
    }
    if ( sortattr != NULL ) {
    extern intstrcasecmp();

    (void) ldap_sort_entries( ld, &res,
    ( *sortattr == '\0' ) ? NULL : sortattr, strcasecmp );
    matches = 0;
    first = 1;
    for ( e = ldap_first_entry( ld, res ); e != NULLMSG;
    e = ldap_next_entry( ld, e ) ) {
matches++;
if ( !first ) {
    putchar( '\n' );
} else {
    first = 0;
}
print_entry( ld, e, attrsonly );
    }
    }

    if ( verbose ) {
        printf( "%d matches\n", matches );
    }

    ldap_msgfree( res );
    return( rc );
}


print_entry( ld, entry, attrsonly )
    LDAP*ld;
    LDAPMessage*entry;
    intattrsonly;
{
    char*a, *dn, *ufn, tmpfname[ 64 ];
```

```
    inti, j, notascii;
    BerElement*ber;
    struct berval**bvals;
    FILE*tmpfp;
    extern char*mktemp();

    dn = ldap_get_dn( ld, entry );
    if ( ldif ) {
write_ldif_value( "dn", dn, strlen( dn ));
    } else {
printf( "%s\n", dn );
    }
    if ( includeufn ) {
ufn = ldap_dn2ufn( dn );
if ( ldif ) {
    write_ldif_value( "ufn", ufn, strlen( ufn ));
} else {
    printf( "%s\n", ufn );
}
free( ufn );
    }
    free( dn );

    for ( a = ldap_first_attribute( ld, entry, &ber ); a != NULL;
    a = ldap_next_attribute( ld, entry, ber ) ) {
if ( skipsortattr && strcasecmp( a, sortattr ) == 0 ) {
    continue;
}
if ( attrsonly ) {
    if ( ldif ) {
write_ldif_value( a, "", 0 );
    } else {
printf( "%s\n", a );
    }
} else if (( bvals = ldap_get_values_len( ld, entry, a )) != NULL ) {
    for ( i = 0; bvals[i] != NULL; i++ ) {
if ( vals2tmp ) {
    sprintf( tmpfname, "/tmp/ldapsearch-%s-XXXXXX", a );
    tmpfp = NULL;

    if ( mktemp( tmpfname ) == NULL ) {
perror( tmpfname );
    } else if (( tmpfp = fopen( tmpfname, "w")) == NULL ) {
perror( tmpfname );
    } else if ( fwrite( bvals[ i ]->bv_val,
```

```
    bvals[ i ]->bv_len, 1, tmpfp ) == 0 ) {
perror( tmpfname );
    } else if ( ldif ) {
write_ldif_value( a, tmpfname, strlen( tmpfname ));
    } else {
printf( "%s%s%s\n", a, sep, tmpfname );
    }

    if ( tmpfp != NULL ) {
fclose( tmpfp );
    }
} else {
    notascii = 0;
    if ( !allow_binary ) {
for ( j = 0; j < bvals[ i ]->bv_len; ++j ) {
    if ( !isascii( bvals[ i ]->bv_val[ j ] )) {
notascii = 1;
break;
    }
}
    }

    if ( ldif ) {
write_ldif_value( a, bvals[ i ]->bv_val,
bvals[ i ]->bv_len );
    } else
 {
printf( "%s%s%s\n", a, sep,
notascii ? "NOT ASCII" : (char *)bvals[ i ]->bv_val );
    }
}
    }
    gsledePBerBvecfree( bvals );
}
    }
}


int
write_ldif_value( char *type, char *value, unsigned long vallen )
{
    char *ldif;

    if (( ldif = gsldlDLdifTypeAndValue( type, value, (int)vallen )) == NULL )
    {
```

```
    return( -1 );
    }

    fputs( ldif, stdout );
    free( ldif );

    return( 0 );
}
```

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).

8

DBMS_LDAP PL/SQL Reference

DBMS_LDAP contains the functions and procedures which can be used by PL/SQL programmers to access data from LDAP servers. This section explains all of the API functions in detail. Be sure that you have read the previous DBMS_LDAP PL/SQL package information before using this section.

This section contains these topics:

Summary of Subprograms

Table 8–1 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/password based 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.
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().

Table 8–1 (Cont.) DBMS_LDAP API Subprograms

Function or Procedure	Description
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 given entry in the result set.
FUNCTION get_values	The function get_values() can be used to retrieve all of the values associated for 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 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 happen after DBMS_LDAP.create_mod_array() is called.
FUNCTION modify_s	Performs a synchronous modification of an existing LDAP directory entry. Before calling `add_s`, we have to call `DBMS_LDAP.creat_mod_array` () and `DBMS_LDAP.populate_mod_array`() first.
FUNCTION add_s	Adds a new entry to the LDAP directory synchronously. Before calling `add_s`, we have to call `DBMS_LDAP.creat_mod_array` () and `DBMS_LDAP.populate_mod_array`() first.
PROCEDURE free_mod_array	Frees the memory allocated by `DBMS_LDAP.create_mod_array`().

Table 8–1 (Cont.) DBMS_LDAP API Subprograms

Function or Procedure	Description
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 for information about the `DBMS_LDAP.search_s()` and `DBMS_LDAP.search_st()` functions
- Terminating the Session by Using DBMS_LDAP for information about the `DBMS_LDAP.unbind_s()` function

Exception Summary

DBMS_LDAP can generate the following exceptions:

Table 8–2 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 local language of the user.
init_failed	31203	Raised by DBMS_LDAP.init() if there are some 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 of the 'search' functions if the scope of the search is invalid.
invalid_search_time_val	31207	Raised by time based search function: DBMS_ LDAP.search_st() if it is given an invalid value for the 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 the functions that modify/add/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.

Table 8–2 (Cont.) DBMS_LDAP Exception Summary

Exception Name	Oracle Error Number	Cause of Exception
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 if the value of the RDN is NULL.
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.
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 one of 1, 2 or 3.

Data-Type Summary

The DBMS_LDAP package uses the following data-types:

Table 8–3 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.

Table 8–3 (Cont.) DBMS_LDAP Data-Type Summary

Data-Type	Purpose
MESSAGE	Used to hold a handle to the message retrieved from the result set. This is used by all functions that work with entries attributes and values.
MOD_ARRAY	Used to hold a handle into the array of modifications being passed into 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 which 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.

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 8–4 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 itself with a colon (:) character. The hosts will be 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 a host includes a port number then this parameter is ignored. If this parameter is not specified and the host name also does not contain the port number, a default port number of 389 is assumed.

Return Values

Table 8–5 INIT Function Return Values

Value	Description
SESSION (function return)	A handle to an LDAP session which can be used for further calls into the API.

Exceptions

Table 8–6 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 in order to establish a session to 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.Subsequent to the call to init(), 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 username/password based authentication to the directory server.

Syntax

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

Parameters

Table 8–7 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 8–8 SIMPLE_BIND_S Function Return Values

Value	Description
PLS_INTEGER (function return)	DBMS_LDAP.SUCCESS on a successful completion. If there was a problem, one of the following exceptions will be raised.

Exceptions

Table 8–9 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 8–10 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
cred	A text string containing the credentials used for authentication
meth	The authentication method

Return Values

Table 8–11 BIND_S Function Return Values

Value	Description
PLS_INTEGER (function return)	DBMS_LDAP.SUCCESS on a successful completion. One of the following exceptions is raised if there was a problem.

Exceptions

Table 8–12 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 SESSION
)
    RETURN PLS_INTEGER;
```

Parameters

Table 8–13 UNBIND_S Function Parameters

Parameter	Description
ld	A valid LDAP session handle.

Return Values

Table 8–14 UNBIND_S Function Return Values

Value	Description
PLS_INTEGER (function return)	DBMS_LDAP.SUCCESS on proper completion. One of the following exceptions is raised otherwise.

Exceptions

Table 8–15 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, will send an unbind request to the server, close all open connections associated with the LDAP session and dispose of all resources associated with the session handle before returning. After a call to this function, the session handle ld is invalid and it is illegal to make any further LDAP API calls using ld.

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 8–16 COMPARE_S Function Parameters

Parameter	Description
ld	A valid LDAP session handle
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 8–17 COMPARE_S Function Return Values

Value	Description
PLS_INTEGER (function return)	COMPARE_TRUE is the given attribute has a matching value. COMPARE_FALSE if the value of the attribute does not match the value given.

Exceptions

Table 8–18 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 if the value of a given attribute stored in the directory server matches a certain value.This operation can only be performed on attributes whose syntax definition allows them to be compared. The compare_s function can only be called after a valid LDAP session handle has been obtained from the init() function and authenticated using 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 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.

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 8–19 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 non-zero if only types are wanted.

Table 8–19 (Cont.) SEARCH_S Function Parameters

Parameter	Description
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 8–20 SEARCH_S Function Return Value

Value	Description
PLS_INTEGER (function return)	DBMS_LDAP.SUCCESS if the search operation succeeded. An exception is raised in all other cases.
res (OUT parameter)	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 8–21 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, values, etc., can be extracted by calling the parsing routines described below.

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 8–22 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.

Table 8–22 (Cont.) SEARCH_ST Function Parameters

Parameter	Description
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 non-zero 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 8–23 SEARCH_ST Function Return Values

Value	Description
PLS_INTEGER (function return)	DBMS_LDAP.SUCCESS if the search operation succeeded. An exception is raised in all other cases.
res (OUT parameter)	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 8–24 SEARCH_ST Function Exceptions

Exception	Description
invalid_session	Raised if the session handle `ld` is invalid.

Table 8–24 (Cont.) SEARCH_ST Function Exceptions

Exception	Description
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 8–25 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 8–26 FIRST_ENTRY Return Values

Value	Description
MESSAGE (function return)	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 8–27 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 8–28 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 8–29 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 8–30 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 8–31 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 8–32 COUNT_ENTRY Function Return Values

Value	Description
PLS INTEGER (function return)	Non-zero if there are entries in the result set -1 if there was a problem.

Exceptions

Table 8–33 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 8–34 FIRST_ATTRIBUTE Function Parameter

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 which attribute in the entry has been read

Return Values

Table 8–35 FIRST_ATTRIBUTE Function Return Values

Value	Description
VARCHAR2 (function return)	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 8–36 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() fetches 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 8–37 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 which attribute in the entry has been read.

Return Values

Table 8–38 NEXT_ATTRIBUTE Function Return Values

Value	Description
VARCHAR2 (function return)	The name of the attribute if it exists.

Exceptions

Table 8–39 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,
    ldapentry IN MESSAGE
)
    RETURN VARCHAR2;
```

Parameters

Table 8–40 GET_DN Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
ldapentry	The entry whose DN is to be returned.

Return Values

Table 8–41 GET_DN Function Return Values

Value	Description
VARCHAR2 (function return)	The X.500 Distinguished name of the entry as a PL/SQL string. NULL if there was a problem.

Exceptions

Table 8–42 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.

Table 8–42 (Cont.) GET_DN Function Exceptions

Exception	Description
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 for 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 8–43 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 8–44 GET_VALUES Function Return Values

Value	Description
STRING_COLLECTION (function return)	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 8–45 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 also 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 '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 8–46 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 8–47 GET_VALUES_LEN Function Return Values

Value	Description
BINVAL_COLLECTION (function return	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 8–48 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 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 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 LDAP Directory Information Tree.

Syntax

```
FUNCTION delete_s
(
    ld      IN SESSION,
    entrydn IN VARCHAR2
)
    RETURN PLS_INTEGER;
```

Parameters

Table 8–49 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 8–50 DELETE_S Function Return Values

Value	Description
PLS_INTEGER (function return)	DBMS_LDAP.SUCCESS if the delete operation wa successful. And exception is raised otherwise.

Exceptions

Table 8–51 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

Table 8–51 (Cont.) DELETE_S Function Exceptions

Exception	Description
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 level entries in the LDAP DIT. A leaf level entry is an entry that does not have any children/ldap entries under it. It 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 8–52 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 non-zero indicates that the attribute values from the old name should be removed from the entry.

Return Values

Table 8–53 MODRDN2_S Function Return Values

Value	Description
PLS_INTEGER (function return)	DBMS_LDAP.SUCCESS if the operation was successful. An exception is raised otherwise.

Exceptions

Table 8–54 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 can achieve the same foundation.

See Also

DBMS_LDAP.rename_s().

FUNCTION err2string

The function err2string() can be used to convert an LDAP error code to string in the local language in which the API is operating

Syntax

```
FUNCTION err2string
(
    ldap_err IN PLS_INTEGER
)
    RETURN VARCHAR2;
```

Parameters

Table 8–55 ERR2STRING Function Parameters

Parameter	Description
ldap_err	An error number returned from one the API calls.

Return Values

Table 8–56 ERR2STRING Function Return Values

Value	Description
VARCHAR2 (function return)	A character string appropriately translated to the local language which describes the error in detail.

Exceptions

Table 8–57 ERR2STRING Function Exceptions

Exception	Description
N/A	None.

Usage Notes

In this release, the exception handling mechanism automatically invokes this if any of the API calls encounter an error.

See Also

N/A

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() or add_s() functions.

Syntax

```
FUNCTION create_mod_array
(
    num IN PLS_INTEGER
)
    RETURN MOD_ARRAY;
```

Parameters

Table 8–58 CREATE_MOD_ARRAY Function Parameters

Parameter	Description
num	The number of the attributes that you want to add/modify.

Return Values

Table 8–59 CREATE_MOD_ARRAY Function Return Values

Value	Description
MOD_ARRAY (function return)	The data structure holds a pointer to an LDAP mod array. NULL if there was a problem.

Exceptions

Table 8–60 CREATE_MOD_ARRAY Function Exceptions

Exception	Description
N/A	No LDAP specific exception will be raised

Usage Notes

This function is one of the preparation steps for DBMS_LDAP.add_s and DBMS_LDAP.modify_s. It is required to call 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 8–61 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 the string values only.

Return Values

Table 8–62 POPULATE_MOD_ARRAY (String Version) Procedure Return Values

Value	Description
N/A	

Exceptions

Table 8–63 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` 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 has to happen after `DBMS_LDAP.create_mod_array()` 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 8–64 POPULATE_MOD_ARRAY (Binary 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

Table 8–64 (Cont.) POPULATE_MOD_ARRAY (Binary Version) Procedure Parameters

Parameter	Description
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

Return Values

Table 8–65 POPULATE_MOD_ARRAY (Binary Version) Procedure Return Values

Value	Description
N/A	

Exceptions

Table 8–66 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 has to happen after `DBMS_LDAP.create_mod_array` called.

See Also

`DBMS_LDAP.create_mod_array()`,`DBMS_LDAP.modify_s()`, `DBMS_LDAP.add_s()`, and `DBMS_LDAP.free_mod_array()`.

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 8–67 MODIFY_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 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 8–68 MODIFY_S Function Return Values

Value	Description
PLS_INTEGER	The indication of the success or failure of the modification operation

Exceptions

Table 8–69 MODIFY_S Function Exceptions

Exception	Description
invalid_session	Invalid LDAP session
invalid_entry_dn	Invalid LDAP entry dn

Table 8–69 (Cont.) MODIFY_S Function Exceptions

Exception	Description
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`, we have to call `DBMS_LDAP.create_mod_array()` and `DBMS_LDAP.populate_mod_array()`.

Syntax

```
FUNCTION add_s
(
    ld      IN DBMS_LDAP.SESSION,
    entrydn IN VARCHAR2,
    modptr  IN DBMS_LDAP.MOD_ARRAY
)
    RETURN PLS_INTEGER;
```

Parameters

Table 8–70 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 8–71 ADD_S Function Return Values

Value	Description
PLS_INTEGER	The indication of the success or failure of the modification operation.

Exceptions

Table 8–72 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 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.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 8–73 FREE_MOD_ARRAY Procedure Parameters

Parameter	Description
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 8–74 FREE_MOD_ARRAY Procedure Return Value

Value	Description
N/A	

Exceptions

Table 8–75 FREE_MOD_ARRAY Procedure Exceptions

Exception	Description
N/A	No LDAP specific exception will be raised.

Usage Notes

N/A

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 8–76 COUNT_VALUES Function Parameters

Parameter	Description
values	The collection of string values.

Return Values

Table 8–77 COUNT_VALUES Function Return Values

Value	Description
PLS_INTEGER	The indication of the success or failure of the operation.

Exceptions

Table 8–78 COUNT_VALUES Function Exceptions

Exception	Description
N/A	No LDAP specific exception will be raised.

Usage Notes

N/A

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 8–79 COUNT_VALUES_LEN Function Parameters

Parameter	Description
values	The collection of binary values.

Return Values

Table 8–80 COUNT_VALUES_LEN Function Return Values

Value	Description
PLS_INTEGER	The indication of the success or failure of the operation.

Exceptions

Table 8–81 COUNT_VALUES_LEN Function Exceptions

Exception	Description
N/A	No LDAP specific exception will be raised.

Usage Notes

N/A

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 8–82 RENAME_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()`.
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 if the old RDN should be retained. If this value is 1, then the old RDN will be removed.
serverctrls	Currently not supported.
clientctrls	Currently not supported.

Return Values

Table 8–83 RENAME_S Function Return Values

Value	Description
PLS_INTEGER	The indication of the success or failure of the operation.

Exceptions

Table 8–84 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.

Usage Notes

N/A

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 8–85 EXPLODE_DN Function Parameters

Parameter	Description
dn	This parameter specifies the name of the directory entry to be broken up.
notypes	This parameter specifies if the attribute tags will be returned. If this value is not 0, then there will be no attribute tags will be returned.

Return Values

Table 8–86 EXPLODE_DN Function Return Values

Value	Description
STRING_COLLECTION	An array of strings. If the DN can not be broken up, NULL will be returned.

Exceptions

Table 8–87 EXPLODE_DN Function Exceptions

Exception	Description
invalid_entry_dn	Invalid LDAP DN.
invalid_notypes	Invalid LDAP notypes value.

Usage Notes

N/A

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 8–88 OPEN_SSL Function Parameters

Parameter	Description
ld	This parameter is a handle to an LDAP session, as returned by a successful call to `DBMS_LDAP.init()`.
sslwrl	This parameter specifies the wallet location (Required for one-way or two-way SSL connection.)
sslwalletpasswd	This parameter specifies the wallet password (Required for one-way or two-way SSL connection.)
sslauth	This parameter specifies the SSL Authentication Mode (1 for no authentication required, 2 for one way authentication required, 3 for two way authentication required.

Return Values

Table 8–89 OPEN_SSL Function Return Values

Value	Description
PLS_INTEGER	The indication of the success or failure of the operation.

Exceptions

Table 8–90 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 passwd.
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 8–91 MSGFREE Function Parameters

Parameter	Description
res	The message handle as obtained by a call to one of the synchronous search routines.

Return Values

Table 8–92 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

N/A. No LDAP-specific exception is raised.

Usage Notes

N/A

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

```
PROCEDURE ber_free
(
    ber_elem IN BER_ELEMENT,
    freebuf  IN PLS_INTEGER
)
```

Parameters

Table 8–93 BER_FREE Function Parameters

Parameter	Description
ber_elem	A handle to BER ELEMENT.
freebuf	The value of this flag should be zero while the BER ELEMENT returned from DBMS_LDAP.first_attribute() is being freed. For any other case, the value of this flag should be one. The default value of this parameter is zero.

Return Values

N/A

Exceptions

N/A. No LDAP-specific exception is raised.

Usage Notes

N/A

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 8–94 Parameters for nls_convert_to_utf8

Parameter	Description
data_local	Contains the database character set data.

Return Values

Table 8–95 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 of UTF8 character set if the UTF8_CONVERSION package variable is set to FALSE. In that case nls_convert_to_utf8() function can be used to convert the input data from database character set to UTF8 character set.

If the UTF8_CONVERSION package variable of DBMS_LDAP package is set to TRUE, then functions in DBMS_LDAP package expect the input data to be of database character set.

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 and returns it.

Syntax

```
Function nls_convert_to_utf8
(
data_local IN STRING_COLLECTION
)
RETURN STRING_COLLECTION;
```

Parameters

Table 8–96 Parameters for nls_convert_to_utf8

Parameter	Description
data_local	Collection of strings containing database character set data.

Return Values

Table 8–97 Return Values for nls_convert_to_utf8

Value	Description
STRING_COLLECTION	Collection of strings containing UTF8 character set data.

Usage Notes

The functions in DBMS_LDAP package expect the input data to be of UTF8 character set if the UTF8_CONVERSION package variable is set to FALSE. In that case nls_convert_to_utf8() function can be used to convert the input data from database character set to UTF8 character set.

If the UTF8_CONVERSION package variable of DBMS_LDAP package is set to TRUE, then functions in DBMS_LDAP package expect the input data to be of 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 data to database character set data and returns it.

Syntax

```
Function nls_convert_from_utf8
(
data_utf8 IN VARCHAR2
)
RETURN VARCHAR2;
```

Parameters

Table 8–98 Parameter for nls_convert_from_utf8

Parameter	Description
data_utf8	Contains the UTF8 character set data.

Return Values

Table 8–99 Return Value for nls_convert_from_utf8

Value	Description
VARCHAR2	Database character set data string.

Usage Notes

The functions in DBMS_LDAP package return UTF8 character set data if the UTF8_CONVERSION package variable is set to FALSE. In that case nls_convert_from_utf8() function can be used to convert the output data from UTF8 character set to database character set.

If the UTF8_CONVERSION package variable of DBMS_LDAP package is set to TRUE, then functions in 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 and returns it.

Syntax

```
Function nls_convert_from_utf8
(
data_utf8 IN STRING_COLLECTION
)
RETURN STRING_COLLECTION;
```

Parameters

Table 8–100 Parameter for nls_convert_from_utf8

Parameter	Description
data_utf8	Collection of strings containing UTF8 character set data.

Return Values

Table 8–101 Return Value for nls_convert_from_utf8

Value	Description
VARCHAR2	Collection of strings containing database character set data.

Usage Notes

The functions in DBMS_LDAP package return UTF8 character set data if the UTF8_CONVERSION package variable is set to FALSE. In that case nls_convert_from_utf8() function can be used to convert the output data from UTF8 character set to database character set.

If the UTF8_CONVERSION package variable of DBMS_LDAP package is set to TRUE, then functions in 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 8–102 Return Value for nls_get_dbcharset_name

Value	Description
VARCHAR2	String containing database character set name.

See Also

DBMS_LDAP.nls_convert_to_utf8(), DBMS_LDAP.nls_convert_from_utf8().

9

DBMS_LDAP_UTL PL/SQL Reference

This chapter contains reference material for the DBMS_LDAP_UTL package, which contains Oracle Extension utility functions. This chapter contains these topics:

Summary of Subprograms

Table 9–1 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 the 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 9–2 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 9–3 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 9–4 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_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.

Function Return Code Summary

The DBMS_LDAP_UTL functions can return the values in the following table

Table 9–5 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 given user doesn't have any group membership.
NO_SUCH_SUBSCRIBER	-4	Returned by subscriber-related functions when the subscriber doesn't exist in the directory.
NO_SUCH_USER	-5	Returned by user-related functions when the user doesn't exist in the directory.
NO_ROOT_ORCL_CTX	-6	Returned by most functions when the root oracle context doesn't 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 doesn't contain all the required information needed by the function.
NO_SUBSCRIBER_ORCL_ CTX	-9	Oracle context doesn't 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 doesn't exist in the directory.
MULTIPLE_GROUP_ ENTRIES	-13	Multiple group entries exist for the given group nickname in the directory.

Table 9–5 Function Return Codes

Name	Return Code	Description
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	Returned by DBMS_LDAP_UTL.authenticate_user() function when the user password needs to be changed. This is a password policy error.
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 doesn't contain at least one numeric character. This is a password policy error.
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.

Table 9–5 Function Return Codes

Name	Return Code	Description
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_userr() 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 9–6 DBMS_LDAP_UTL Data Types

Data Type	Purpose
HANDLE	Used to hold entity related.
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.

User-Related Subprograms

A user is represented using DBMS_LDAP_UTL.HANDLE data type. You can create a user 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 user. Here is an example of a user handle creation:

```
  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 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_user_handle_properties(
        user_handle,
        DBMS_LDAP_UTL.SUBSCRIBER_HANDLE,
        subscriber_handle
        );
```

Some common usage of User handles include setting and getting user properties, and authentication of the user. Here is an example of authenticating 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 an example of getting the telephone number of the user:

```
--    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-14 for samples of user handle

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 9–7 AUTHENTICATE_USER Function Parameters

Parameter Name	Parameter Type	Parameter Description
ld	SESSION	A valid LDAP session handle.
user	HANDLE	The user handle.
auth_type	PLS_INTEGER	Type of authentication. Valid values are as follows: - DBMS_LDAP_UTL.AUTH_SIMPLE
credentials	VARCHAR2	The user credentials. Valid values are as follows: for DBMS_LDAP_UTL.AUTH_SIMPLE - password
binary_ credentials	RAW	The binary credentials. Valid values are as follows: for DBMS_LDAP_UTL.AUTH_SIMPLE - NULL

Return Values

Table 9–8 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 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.INVALID_SUBSCRIBER_ORCL_CTX	Invalid Subscriber Oracle Context.
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.INVALID_ROOT_ORCL_CTX	Invalid Root Oracle Context.
DBMS_LDAP_UTL.ACCT_TOTALLY_LOCKED_EXCP	User account is locked.
DBMS_LDAP_UTL.AUTH_PASSWD_CHANGE_WARN	Password should be changed.
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	Returns 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.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 9–9 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 9–10 CREATE_USER_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_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 9–11 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 9–12 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 doesn't need 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 9–13 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 attributes to fetch for the user.
ptype	PLS_INTEGER	Type of properties to return. Valid values are as follows: - DBMS_LDAP_UTL.ENTRY_PROPERTIES - DBMS_LDAP_UTL.NICKNAME_PROPERTY
ret-pset_ collection	PROPERTY_ SET_ COLLECTION	The user details containing the attributes requested by the caller.

Return Values

Table 9–14 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 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.INVALID_ROOT_ORCL_CTX	Invalid Root Oracle Context.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.
DBMS_LDAP error codes	Returns 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 user type is of: - DBMS_LDAP_UTL.TYPE_NICKNAME.

This function doesn't 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 any of the following, then the subscriber handle doesn't need to be set in the user handle properties:

- DBMS_LDAP_UTL.TYPE_GUID

- DBMS_LDAP_UTL.TYPE_DN .

If the subscriber handle is set, then it would be 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 9–15 SET_USER_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter 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. Valid values are as follows: - ENTRY_PROPERTIES
mod_pset	PROPERTY_ SET	Data structure containing modify operations to perform on the property set.
mod_op	PLS_INTEGER	The type of modify operation to be performed on the property set. Valid values are as follows: - ADD_PROPERTYSET - MODIFY_PROPERTYSET -DELETE_PROPERTYSET

Return Values

Table 9–16 SET_USER_PROPERTIES Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
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.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	Returns 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,
ptype IN PLS_INTEGER,
filter IN VARCHAR2,
rep_pset_coll OUT PROPERTY_SET_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

Table 9–17 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. Valid values are as follows: - 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 9–18 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.

Value	Description
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.
USER_PROPERTY_NOT_FOUND	User extended property doesn't exist.
DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX	Invalid Root Oracle Context.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.
DBMS_LDAP error codes	Returns 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_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 9–19 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 9–20 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 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.INVALID_ROOT_ORCL_CTX	Invalid Root Oracle Context.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.
DBMS_LDAP error codes	Returns 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().

Function check_group_membership

The function check_group_membership() checks the membership of the user 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 9–21 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. Valid values are as follows: DBMS_LDAP_UTL.NESTED_MEMBERSHIP DBMS_LDAP_UTL.DIRECT_MEMBERSHIP

Return Values

Table 9–22 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 only be called 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 9–23 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 9–24 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.

Value	Description
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	Returns 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.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 9–25 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. Valid values are as follows: 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 9–26 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 only be called 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 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 usage 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 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: Group-Related Functions on page B-27 for more usage samples of group handle

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 9–27 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

Parameter Name	Parameter Type	Parameter Description
group_id	VARCHAR2	The group ID representing the group entry.

Return Values

Table 9–28 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 9–29 SET_GROUP_HANDLE_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
group_hd	HANDLE	A pointer to the handle to the group.

Parameter Name	Parameter Type	Parameter Description
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 9–30 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 9–31 GET_GROUP_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
ld	SESSION	A valid LDAP session handle.
group_handle	HANDLE	The group handle.
attrs	STRING_ COLLECTION	A list of attributes that must be fetched for the group.
ptype	PLS_INTEGER	The type of properties to be returned. Valid values are as follows: - DBMS_LDAP_UTL.ENTRY_PROPERTIES
ret_pset_coll	PROPERTY_ SET_ COLLECTION	The group details containing the attributes requested by the caller.

Return Values

Table 9–32 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	Returns 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 doesn't 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 any of the following, then the subscriber handle doesn't need to be set in the group handle properties:

- DBMS_LDAP_UTL.TYPE_GUID

- DBMS_LDAP_UTL.TYPE_DN .

If the subscriber handle is set, then it would be 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 9–33 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 9–34 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	Returns 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().

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 a simple name. When a simple name is used, additional information from the root Oracle Context is used to identify the subscriber. Here is an example of a subscriber handle creation:

```
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`.

A common usage of subscriber handle is getting subscriber properties. 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-14 for samples of subscriber handle

Function create_subscriber_handle

The function create_subscriber_handle() creates a subscriber handle.

Syntax

```
FUNCTION create_subscriber_handle
(
ld IN SESSION,
subscriber_hd OUT HANDLE,
subscriber_type IN PLS_INTEGER,
subscriber_id IN VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters

Table 9–35 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 as follows: - DBMS_LDAP_UTL.TYPE_DEFAULT Then the default subscriber is fetched from Root Oracle Context.

Return Values

Table 9–36 CREATE_SUBSCRIBER_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_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 9–37 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 fetched for the subscriber.
ptype	PLS_INTEGER	The type of properties to return. Valid values are as follows: - DBMS_LDAP_UTL.ENTRY_PROPERTIES - DBMS_LDAP_UTL.COMMON_PROPERTIES, to retrieve the subscriber's Oracle Context Properties.
ret_pset_coll	PROPERTY_ SET_ COLLECTION	The subscriber details containing the attributes requested by the caller.

Return Values

Table 9–38 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.

Value	Description
DBMS_LDAP_UTL.MULTIPLE_SUBSCRIBER_ENTRIES	Multiple number of subscriber DN entries exist in the directory for the given subscriber.
DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX	Invalid Root Oracle Context.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.
DBMS_LDAP error codes	Returns 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.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 9–39 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 9–40 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.
DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX	Invalid Root Oracle Context.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.
DBMS_LDAP error codes	Returns 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().

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 9–41 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 attributes to fetch for the subscriber.
ptype	PLS_INTEGER	The type of properties to return. Valid values are as follows: - DBMS_LDAP_UTL.DEFAULT_RAD_PROPERTIES
filter	VARCHAR2	An LDAP filter to further refine the subscriber properties returned by the function.

Table 9–41 (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 9–42 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 doesn't exist.
DBMS_LDAP_UTL.INVALID_ROOT_ ORCL_CTX	Invalid Root Oracle Context.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.
DBMS_LDAP error codes	Returns proper DBMS_LDAP error codes for unconditional failures while carrying outLDAP 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_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 sample usage illustrating 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: Property-Related Subprograms on page B-19 for more usage samples of the Property-related subpropgrams

Miscellaneous Subprograms

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 9–43 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 9–44 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 9–45 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 9–46 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 9–47 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 9–48 GET_PROPERTY_VALUES Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.

Value	Description
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 9–49 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 9–50 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 9–51 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()

Return Values

N/A

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,
)
RETURN PLS_INTEGER;
```

Parameters

Table 9–52 CREATE_MOD_PROPERTYSET Function Parameters

Parameter Name	Parameter Type	Parameter Description
pset_type	PLS_INTEGER	The type of property set being modified. Valid values are as follows: - 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 9–53 CREATE_MOD_PROPERTYSETFunction Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
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 9–54 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. Valid values are as follows: - 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 9–55 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 9–56 FREE_MOD_PROPERTYSET Procedure Parameters

Parameter Name	Parameter Type	Parameter Description
mod_pset	PROPERTY_SET	Mod_PropertySet data structure.

Return Values

N/A

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 9–57 FREE_HANDLE Procedure Parameters

Parameter Name	Parameter Type	Parameter Description
handle	HANDLE	A pointer to a handle.

Return Values

N/A

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 for support of the interface version.

Syntax

```
FUNCTION check_interface_version
(
interface_version IN VARCHAR2
)
RETURN PLS_INTEGER;
```

Parameters

Table 9–58 CHECK_INTERFACE_VERSION Function Parameters

Parameter Name	Parameter Type	Parameter Description
interface_ version	VARCHAR2	Version of the interface.

Return Values

Table 9–59 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 Return Code Summary

The DBMS_LDAP_UTL functions can return the values in the following table

Table 9–60 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 given user doesn't have any group membership.
NO_SUCH_SUBSCRIBER	-4	Returned by subscriber-related functions when the subscriber doesn't exist in the directory.
NO_SUCH_USER	-5	Returned by user-related functions when the user doesn't exist in the directory.

Table 9–60 Function Return Codes

Name	Return Code	Description
NO_ROOT_ORCL_CTX	-6	Returned by most functions when the root oracle context doesn't 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 doesn't contain all the required information needed by the function.
NO_SUBSCRIBER_ORCL_ CTX	-9	Oracle context doesn't 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 doesn't 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	Returned by DBMS_LDAP_UTL.authenticate_user() function when the user password needs to be changed. This is a password policy error.
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.

Table 9–60 Function Return Codes

Name	Return Code	Description
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 doesn't contain at least one numeric character. This is a password policy error.
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_userr() 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 9–61 DBMS_LDAP_UTL Data Types

Data Type	Purpose
HANDLE	Used to hold entity related.
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.

10

DAS_URL Interface Reference

This chapter describes the Oracle extensions to the DAS_URL Service Interface. It contains these sections:

Oracle Delegated Administration Services Units and Corresponding Directory Entries

Table 10–1 lists each Oracle Delegated Administration Services unit and the corresponding entry in Oracle Internet Directory which stores the relative URL.

Table 10–1 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`
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`

Table 10–1 (Cont.) Service Units and Corresponding Entries

Service Unit	Entry
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

The following table lists all the available DAS units and the URL parameters that can be passed to DAS units.

Table 10–2 DAS Units and Corresponding URL Parameters

DAS Unit	Parameter	Return Values
Create User	`homeURL, doneURL, cancelURL,enablePA`	`returnGUID`

Table 10–2 (Cont.) DAS Units and Corresponding URL Parameters

DAS Unit	Parameter	Return Values
Edit User	homeURL, doneURL cancelURL, enablePA	
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, cancelURLenablePA , groupGUID	
DeleteGroup	homeURL, doneURL, cancelURL	
DeleteGroupGivenGUID	homeURL, doneURL, cancelURL, groupGUID	
GroupPrivilege	homeURL, doneURL, cancelURL	
GroupPrivilegeGivenGUID	homeURL, doneURL, cancelURL, groupGUID	
AccountInfo	homeURL, doneURL, cancelURL	

Table 10–2 (Cont.) DAS Units and Corresponding URL Parameters

DAS Unit	Parameter	Return Values
PasswordChange	homeURL, doneURL, cancelURL	
UserSearch	homeURL, doneURLm, cancelURL	
GroupSearch	homeURL, doneURL, cancelURL	
UserLOV	base, cfilter, title dasdomain	
GroupLOV	otype, base, cfilter, title dasdomain	

DAS URL API Parameter Descriptions

The following parameters are used with DAS units.

Table 10–3 DAS URL Parameter Descriptions

Parameter	Description
homeURL	The URL which is linked to the global button Home. When the calling application specifies this value, clicking the Home button will redirect 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 case of Create User, once the user is created, clicking OK will redirect the URL to this location. Hence the user navigation experience will be smooth.
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/false. This will enable the section Assign Privileges in User or Group operation. If the enablePA is passed with value of true in the Create User page, then the Assign Privileges to User section will also appear in the Create User page.
userGUID	This is the GUID of the user to be edited or deleted. This corresponds to the orclguid attribute. Specifying this will skip the search for the User step in either editUser or deleteUser units.

Table 10–3 (Cont.) DAS URL Parameter Descriptions

Parameter	Description
GroupGUID	This is the GUID of the group to be edited or deleted. This corresponds to the orclguid attribute. Specifying this will skip the search for the group step in either editGroup or deleteGroup units.
parentDN	When this parameter is specified in CreateGroup, the group will be created under this container. If not specified, group creation will default to the group search base.
base	This parameter represents the search base in 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 doneURL 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 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/`

User or Group List of Values Access

In DAS, the search page for users or groups is called the List of Values (LOV).

DAS uses Java scripts to access the LOV, and to pass values between the LOV calling window and DAS LOV page. Since the Java scripts have security restrictions, data cannot pass across the domains. Due to this limitation, only the pages in the same domain can access the DAS LOV units.

The example below is a simple HTML file which invokes the DAS User LOV. Note the Javascript functions are taken from the `MarlinCore.js` file published with UIX. Applications may use their own Javascript procedures or use the UIX library utilities.

```
<html>
 <head>
  <title>test</title>
```

```
 </head>
 <script src="/cabo/jsLibs/MarlinCore.js">
 </script>
 <body>
  <form name="form1" method="GET" action="test.html">
    <script language="javascript">
       function doSelect(lovWindow, event)
       {
            var value = lovWindow.userName;
             if (value!=(void 0)){
                 document.form1.lov1.value = lovWindow.userName;
                 document.form1.lov2.value = lovWindow.userGuid
             }
        }
    </script>
 <table border="0" cellspacing="0" cellpadding="0">
<tr>
  <td align="right" nowrap>
   <span> 
     <label for="M__Id2500">Manager</label>
    </span>
  </td>
<td nowrap>
  <span>
    <input id="M__Id2500" name="lov1" size="25" type="text">
     <a onclick="openWindow(top,
'/oiddas/cabo/jsps/frameRedirect.jsp?redirect=http://sneaker.us.oracle.com:7777/
oiddas/ui/oracle/ldap/das/search/LOVUserSearch?title=User',
'lovWindow',{width:600,height:600},
false, 'dialog',doSelect);
return false;"
href="">
                            <img src="/oiddas/oracle/ldap/das/Images/c-lov.gif"
   alt="Testing lov" border="0" align="absmiddle">
     </a>
   </span>
  </td>
 </tr>
 <tr>
     <td align="right" nowrap>
       <span> 
            <label for="M__Id2501">Guid</label>
       </span>
     </td>
     <td>
```

```
        <input id="M__Id2501" name="lov2" size="25" type="text">
      </td>
  </tr>
 </table>
 </form>
 </body>
</html>
```

More details about the `UIX openWindow` function are available at:

`http://cabo/cabo/marlin/docs/windowJS.html`

11

Provisioning Integration API Reference

This chapter contains reference information for the Oracle Directory Provisioning Integration Service Registration API. It contains the following sections:

Versioning of Provisioning Files and Interfaces

In the Oracle Internet Directory release 9.0.2, the default interface version was version 1.1. In release 9.0.4, the interface version defaults to version 2.0, but the administrator can set this back to version 1.1 to maintain the previous interface.

Extensible Event Definition Configuration

This feature is meant only for OUTBOUND events. This feature 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 the 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. E.g The object could be a USER, GROUP, IDENTITYetc.
- LDAP Change Type (`orclODIPProvEventChangeType`): This indicates what 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 qualifies an LDAP entry to be of a specific Object Type. For example, `Objectclass=orclUserV2` means that any LDAP entry which satisfies this criteria can be qualified as this Object Type and any change to this entry can generate appropriate event(s).

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 11–1 lists the event definitions predefined as a part of the installation.

Table 11–1 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, DELETE, MODIFY	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 OID. The DIP server would recognize this new EVENT definition and propagate 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/deleted, DIP will propagate the XYZ_ADD/XYZ_ MODIFY/XYZ_DELETE event respectively 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 11–2 on page 11-6.

Table 11–2 Attributes of the Provisioning Subscription Profile

Attribute	Description
EventSubscriptions	OUTBOUND Events only. (Multi valued) This is same as it was in the previous release. Events for which DIP should send notification to this application. Format of this string :"[USER]GROUP]:[Domain of interest>]:[DELETE | ADD | MODIFY(<list of attributes separated by comma>)]" Multiple values may be specified by listing the parameter multiple times each with different values. If not specified the following defaults are assumed: USER:<org. DN>:DELETEGROUP:<org. DN>:DELETE—that is, send user and group delete notifications under the organization DN.
MappingRules	INBOUND Events Only (Multi valued) New to this release This 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. OBJECT_TYPE: Filter condition: Domain Of Interest Multiple rules are allowed. For example: ■ EMP::cn=users,dc=acme,dc=com This means that if the object type received is "EMP", the event is meant for the domain "cn=users,dc=acme,dc=com" ■ EMP:l=AMERICA:l=AMER,cn=users,dc=acme,dc=com This means that if the object type received is "EMP", and the event has the attribute l (locality) and its value is "AMERICA", the event is meant for the domain "l=AMER,cn=users,dc=acme,dc=com"
permittedOperations	INBOUND Events Only (Multi valued) New to this release. This is used to define the types of EVENT an application is privileged to send to the Provisioning Integration Service. Format : Event_Object: Affected Domain:Operation(Attributes,…) For example: ■ IDENTITY:cn=users,dc=acme,dc=com:ADD(*) This means that IDENTITY_ADD event is allowed for the specified domain and all attributes are also allowed. ■ IDENTITY:cn=users,dc=acme,dc=com:MODIFY(cn,sn.mail,telephonenumber) This means that IDENTITY_MODIFY is allowed for only the attributews in the list. Any extra attributes are silently ignored.

PL/SQL Bidirectional Interface (Version 2.0)

The PL/SQL callback interface requires you to develop a PL/SQL package that Oracle 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 by 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 4-20, you must develop logic to consume events generated by the Oracle Directory Provisioning Integration Service.The PL/SQL callback interface requires you to develop a PL/SQL package that Oracle Directory 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 by 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 a notifier interface implemented by Provisioning System
--                clients to receive information about changes in OID.
--                The name of package can be customized as needed.
--                The functions 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/receive information to/from OID. The name of
-- Package can be customized as needed.
-- The functions 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 Dispostions Constants

```
EVENT_SUCCESS              CONSTANT VARCHAR2(32)  := 'EVENT_SUCCESS';
EVENT_FAILURE              CONSTANT VARCHAR2(32)  := 'EVENT_FAILURE';
EVENT_RESEND               CONSTANT VARCHAR2(32)  := 'EVENT_RESEND';
```

Callbacks

A callback function invoked by the Oracle Directory 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 un-normalized form—that is, if an attribute has two values then two rows would be sent in the collection.

GetAppEvent()

The Oracle directory integration and provisioning server invokes this API in the remote database. It is up to the appliction to respond with an event. Once the Oracle Directory Integration and Provisioning platform gets the event, it processes the it 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—then it responds with `EVENT_RESEND` and the application is expected to resend that event in the future when `GetAppEvent()` is invoked again.

If the provisioning server is able to process the event, but it 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. It is not required to resend the event. 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 reaons.

PutAppEventStatus()

The Oracle directory integration and provisioning server invokes this callback in the remote database after processing an event it had received using the GetAppEvent() callback. For every event received, the Oracle directory integration and provisioning server sends the status event back after processing the event.

```
PROCEDURE PutAppEventStatus (event_status IN LDAP_EVENT_STATUS);
```

PutOIDEvent()

The Oracle directory integration and provisioning server invokes this API in the remote database. It sends event to applications using this callback. It also expects n status event object in response as an OUT parameter. If valid event status object is not sent back or it indicates a RESEND, then the Oracle directory integration and provisioning server resends this event again. 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 explains 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, "Sample Usage"

A

Syntax for LDIF and Command-Line Tools

This appendix provides syntax, usage notes, and examples for **LDAP Data Interchange Format (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 a forward slash ("/").
- 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 Documentation" on page xxvi 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 issue commands through OID Control Utility (OIDCTL) to start or stop directory server instances, your commands are interpreted by this process.

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.

Table A–2 (Cont.) Arguments for Stopping OID Monitor

Argument	Description
`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 OIDCTL 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

Use the **OID Control Utility** 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
[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' -server number_of_server_processes] 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

Table A–3 (Cont.) Arguments for Starting a Directory Server by Using OIDCTL

Argument	Description
`-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` *number_of_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` *maximum_number_of_worker_threads_per_server*	Specifies the maximum number of worker threads for this server
`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

Table A–3 (Cont.) Arguments for Starting a Directory Server by Using OIDCTL

Argument	Description
`instance=`*`server_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=`*`server_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_server_port_number`*	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).

Table A–4 (Cont.) Arguments for Starting a Directory Replication Server by Using

Argument	Description
`-d debug_level`	Specifies a debug level during replication server instance startup
`-h directory_server_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-4.

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]
   [debug=debug_level] [refresh=interval_between_refresh]
   [grpID=group_identifier_of_provisioning_profile]
   [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.

Table A–5 (Cont.) Description of Arguments for Starting the Oracle Directory Integration and Provisioning Server

Argument	Description
`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=`*`configuration_set_number`*	Specifies the number of the configuration set that the directory integration and provisioning server is to execute. This argument is mandatory.
`host=`*`hostname`*	Oracle directory server host name
`port=`*`port_number`*	Oracle directory server port number
`debug=`*`debug_level`*	The required debugging level of the directory integration and provisioning server ▪ **See Also:** The chapter on "Logging, Auditing, and Monitoring the Directory" in in *Oracle Internet Directory Administrator's Guide* for a description of the various debug levels
`refresh=`*`interval_between_refreshes`*	Specifies the interval, in minutes, between server refreshes for any changes in the integration profiles. Default is 2 minutes (Refresh=2).
`maxprofiles=`*`number_of_profiles`*	Specifies the maximum number of profiles that can be executed concurrently for this server instance

Table A–5 (Cont.) Description of Arguments for Starting the Oracle Directory Integration and Provisioning Server

Argument	Description

`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 Oracle Internet 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-13 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

The way you stop the directory integration and provisioning server depends on the tool that you used to start it.

Stopping the Oracle Directory Integration and Provisioning Server by Using OID Monitor and the OID Control Utility If you started the directory integration and provisioning server by using OID Monitor and the OID Control utility, then you use them to stop it, as follows:

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-4.

2. Stop the directory integration and provisioning server by entering:

   ```
   oidctl [connect=connect_string] server=odisrv instance=instance stop
   ```

Stopping the Oracle Directory Integration and Provisioning Server Without Using 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 Control. To stop the server without these tools, use the `stopodiserver.sh` tool, which is located in the *`$ORACLE_HOME`*`/ldap/admin` 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: `http://sources.redhat.com`
- MKS Toolkit 6.1. Visit: `http://www.datafocus.com/`

See Also: "The StopOdiServer.sh Tool Syntax" on page A-62 for instructions about using the stopodiserver.sh tool

Note: If the Oracle directory integration and provisioning server is stopped by any means other than the methods mentioned in this section, then the server cannot be started from the same host. In that case, the footprint of the previous execution in the directory needs to be removed by using the following command:

```
$ORACLE_HOME/ldap/admin/stopodiserver.sh [-host
directory_server_host] [-port directory_server_
port] [-binddn super_user_dN (default is
cn=orcladmin)] [-bindpass super_user_password
(default is welcome)] -instance number_of_the_
instance_to_stop -clean
```

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 Oracle Internet Directory Servers on Either a Virtual Host or a Rack Node

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 on Either a Virtual Host or a Rack Node

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 on Either a Virtual Host or a Rack Node

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 Oracle Directory Integration and Provisioning Server on Either a Virtual Host or a Rack Node

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 cold failover configuration.

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
- ldapaddmt 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: `http://sources.redhat.com`
- MKS Toolkit 6.1. Visit: `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-6 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-4 for information about starting OID Monitor
- The section "Matching Rules"in *Oracle Internet Directory Administrator's Guide* for the matching rules supported by Oracle Internet Directory

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-33.

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-2.

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.

Table A–7 (Cont.) Arguments for ldapadd

Optional Arguments	Description
`-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*—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 Appendix G, "Globalization Support in the Directory"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-2.
`-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.
`-p directory_server_port_number`	Connects 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).
`-P wallet_password`	Specifies wallet password required for one-way or two-way SSL connections

Table A–7 (Cont.) Arguments for ldapadd

Optional Arguments	Description
`-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-2.

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 Appendix G, "Globalization Support in the Directory"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).

Table A–8 (Cont.) Arguments for ldapaddmt

Optional Arguments	Description
`-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"
`-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 Appendix G, "Globalization Support in the Directory"in *Oracle Internet Directory Administrator's Guide*.

Table A–9 (Cont.) Arguments for ldapbind

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.
-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_security_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–10 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 on "Logging, Auditing, and Monitoring the Directory" in *Oracle Internet Directory Administrator's Guide*.
-E "*character_set*"	Specifies native character set encoding. See Appendix G, "Globalization Support in the Directory"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.
-O *ref_hop_limit*	Specifies the number of referral hops that a client should process. The default value is 5.

Table A–10 Arguments for ldapcompare

Optional Arguments	Description
-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–11 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.
-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 Appendix G, "Globalization Support in the Directory"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

Table A–11 (Cont.) Arguments for ldapdelete

Optional 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*	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–12 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*—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 Appendix G, "Globalization Support in the Directory"in *Oracle Internet Directory Administrator's Guide*.
-f *file_name*	Specifies the input file name

Table A–12 Arguments for ldapmoddn

Argument	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.
-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-2.

The list of arguments in the following table is not exhaustive. These arguments are all optional.

Table A–13 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 Appendix G, "Globalization Support in the Directory"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.

Table A–13 (Cont.) Arguments for ldapmodify

Argument	Description
-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*	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-2) 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-2).

- `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]
```

For example:

```
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`, then 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 in the attribute definition entry in the LDIF file the keyword SINGLE-VALUE with surrounding 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-2.

> **See Also:** "ldapmodify Syntax" on page A-33 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–14 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.)

Table A–14 (Cont.) Arguments for ldapmodifymt

Argument	Description
`-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 Appendix G, "Globalization Support in the Directory"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
`-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.

Table A–14 (Cont.) Arguments for ldapmodifymt

Argument	Description
`-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–15 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
-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 on "Logging, Auditing, and Monitoring the Directory" in *Oracle Internet Directory Administrator's Guide*)
-E "*character_set*"	Specifies native character set encoding. See Appendix G, "Globalization Support in the Directory"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

Table A–15 (Cont.) Arguments for ldapsearch

Argument	Description
-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
-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

Table A–16 lists the tasks you can perform by using the Directory Integration and Provisioning Assistant and the corresponding commands. It also points you to instructions for performing each task.

Table A–16 Summary of Functionality of the Directory Integration and Provisioning Assistant

Tasks	Commands	More Information

Create, modify, or delete a synchronization profile	`createprofile`
	`modifyprofile`
	`deleteprofile`
		"Creating, Modifying, and Deleting Synchronization Profiles" on page A-47

See all the profile names in Oracle Internet Directory	`listprofiles`	"Listing All Synchronization Profiles in Oracle Internet Directory" on page A-54
See the details of a specific profile	`showprofile`	"Viewing the Details of a Specific Synchronization Profile" on page A-54
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-49
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-55
Reset the password of the administrator of the Oracle Directory Integration Platform	`chgpasswd`	"Changing the Password of the Administrator of the Oracle Directory Integration and Provisioning Platform" on page A-54
Move integration profiles from one identity management node to another	`reassociate`	"Moving an Integration Profile to a Different Identity Management Node" on page A-55

The command-line interface for the Directory Integration and Provisioning Assistant is:

dipassistant *command* [-help]

command := *Directory Integration and Provisioning Assistant command*

```
Directory Integration and Provisioning Assistant command :=
       createprofile [cp]
       | modifyprofile [mp]
       | deleteprofile [dp]
       | listprofiles[lsprof]
       | showprofile[sp]
       | bootstrap [bs]
       | wpasswd [wp]
       | chgpasswd [cpw]
```

```
        | reassociate [rs]
```

For help on a particular command, enter:

```
dipassistant command -help
```

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 | modifyprofile | deleteprofile
[-host host name] [-port port number] [-dn bind_DN] [-passwd password]
{-file file name | -profile profile name } [propName1=value]
[propName2=value]... [-configset configset_number]
```

For example:

```
dipassistant createprofile -host myhost -port 3060 -passwd xxxx
-file import.profile -configset 1

dipassistant modifyprofile -host myhost -port 3060 -passwd xxxx
-file import.profile -dn xxxx -passwd xxxx -profile myprofile
[propName1=value]
[propName2=value]...

dipassistant deleteprofile -profile myprofile [-host myhost] [-port 3060] [-dn
xxxx] [-passwd xxxx] [-configset 1]
```

Table A–17 on page A-47 describes the parameters for creating, modifying, and deleting synchronization profiles by using the Directory Integration and Provisioning Assistant.

Table A–17 Parameters for Creating, Modifying, and Deleting Synchronization Profiles by Using the Directory Integration and Provisioning Assistant

Parameter	Description
-host	Host where Oracle Internet Directory is running. The default value is the name of the local host.
-port	Port at which Oracle Internet Directory was started. The default is 389.
-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.

Table A–17 (Cont.) Parameters for Creating, Modifying, and Deleting Synchronization Profiles by Using the Directory Integration and Provisioning Assistant

Parameter	Description
`-passwd`	The password of the bind DN to be used while binding to the directory.
`-file`	The file containing all the profile parameters. **See Also:** Table A–18 on page A-48 for a list of parameters and their description
`-configset`	Number of the configuration set entry with which the profile needs to be associated
`-profile`	Profile that needs to modified

The properties expected by `createprofile` and `modifyprofile` commands are described in Table A–18. When modifying an already existing profile, no defaults are assumed. Only those attributes specified in the file are changed.

Table A–18 Properties Expected by createprofile and modifyprofile Commands

Parameter	Description	Default
`odip.profile.name`	Name of the profile	-
`odip.profile.password`	Password for accessing this profile	-
`odip.profile.status`	Either `DISABLE` or `ENABLE`	`DISABLE`
`odip.profile.syncmode`	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
`odip.profile.retry`	Maximum number of times this profile should be executed in the case of an error before the integration server gives up	4
`odip.profile.schedinterval`	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.	1 Minute
`odip.profile.agentexecommand`	In the case of a NON-LDAP interface, the command to produce the information in LDIF format	-
`odip.profile.condirurl`	Location of third-party directory [`hostname:port`]	-

Table A–18 (Cont.) Properties Expected by createprofile and modifyprofile Commands

Parameter	Description	Default
odip.profile.condiraccount	DN or user name used to connect to the third party directory.	-
odip.profile.condirpassword	Password used for identification to the third-party directory.	-
odip.profile.interface	Indicator as to whether the LDAP or LDIF or DB or TAGGED format is to be used for data exchange	LDAP
odip.profile.configfile	Name of the file that contains the additional profile-specific information to be used for execution	-
odip.profile.mapfile	Name of the file that contains the mapping rules	-
odip.profile.condirfilter	Filter that needs to be applied to the changes read from the connected directory before importing to Oracle Internet Directory	-
odip.profile.oidfilter	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.lastchgnum	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	-

Bootstrapping a Directory by Using the Directory Integration and Provisioning Assistant

The command-line interface to the bootstrap command is:

```
dipassistant bootstrap { -profile profile_name [-host host_name] [-port port_
number] -dn bind_DN [-passwd password] [-log log_file] [-logseverity severity]
[-trace trace_file] [-tracelevel trace_level] [-loadparallelism <#nThrs>]
[-loadretry <retryCnt>] | -cfg file_name }
```

For example, either:

```
dipassistant bs -cfg bootstrap cfg
```

or

```
dipassistant bs -host myhost -port 3060 -dn cn=orcladmin -password xxxx -profile
iPlanetProfile
```

Table A–19 Parameters of a deleteprofile Command

Parameter	Description
`-cfg`	A configuration file containing all the parameters required for performing the bootstrapping. **See Also:** Table A–20 on page A-51 for a list of parameters and their description
`-host`	Host where Oracle Internet Directory is running
`-port`	Port at which Oracle Internet Directory was started
`-dn`	The Bind Dn to be used in identifying to the directory
`-password`	The password of the Bind DN to be used while binding to the directory
`-profile`	The profile name.
`-log`	Log file. If this parameter is not specified, then, by default, the log information is written to `OH/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 purpose
`-trace level`	Trace level
`-loadRetry`	When the loading to the destination fails, the number of times the retry should be made before marking the entry as bad entry
`-loadparallelism`	Indicator that loading to Oracle Internet Directory is to take place in parallel by using multiple threads. For example, `-loadparallelism 5` means that 5 threads are to be created, each of which tries to load the entries in parallel to Oracle Internet Directory.

Properties Expected by the Bootstrapping Command

Table A–20 Bootstrapping 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

Table A–20 (Cont.) Bootstrapping Properties

Property	Description	Mandatory	Default
`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`
`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	-

Table A–20 (Cont.) Bootstrapping Properties

Property	Description	Mandatory	Default
`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
`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	`$`*`ORACLE_HOME`*`/ldap/odi/log/bootstrap.trc`
`odip.bootstrap.trclevel`	The tracing level	No	3

Changing the Password of the Administrator of the Oracle Directory Integration and Provisioning Platform

The default password for the `dipadmin` account is same as `ias_admin` password chosen during installation. This command lets you reset the password of `dipadmin` account. To reset that password, you must provide the security credentials of the `orcladmin` account.

For example:

```
$ dipassistant chgpasswd -passwd orcladmin password -host oid.heman.com
-port 3060
```

The Assistant then prompts for the new password as follows:

```
New Password:
Confirm Password:
```

Listing All Synchronization Profiles in Oracle Internet Directory

The listprofiles command prints a list of all the synchronization profiles in Oracle Internet Directory. For example:

```
$ dipassistant listprofiles -passwd dipadmin password -host oid.heman.com
-port 3060
```

This command prints the following sample list:

```
IplanetExport
IplanetImport
ActiveImport
ActiveExport
LdifExport
LdifImport
TaggedExport
TaggedImport
OracleHRAgent
ActiveChgImp
```

Note: The list shown here is the default set of profiles created during installation.

Viewing the Details of a Specific Synchronization Profile

The showprofile command prints the details of a specific synchronization profile For example:

```
$ dipassistant showprofile -passwd dipadmin password -host oid.heman.com
-port 3060 -profile ActiveImport
```

This command prints the following sample output:

```
odip.profile.version = 1.0
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.retry = 5
odip.profile.debuglevel = 0
odip.profile.status = DISABLE
```

Setting the Wallet Password for the Oracle Directory Integration and Provisioning Server

The *WPasswd* 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.

Moving an Integration Profile to a Different Identity Management Node

You can use 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.

Table A–21 describes the reassociation rules.

Table A–21 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 usage is as follows

```
dipassistant reassociate [-src_ldap_host <hostName>]
[-src_ldap_port <portNo>] [-src_ldap_dn <bindDn>] [-src_ldap_passwd
<password>] -dst_ldap_host <hostName> [-dst_ldap_port <portNo>]
[-dst_ldap_dn <bindDn>] [-dst_ldap_passwd <password>] [-log <logfile>]
Options:
-src_ldap_host <hostName> : Host where OID-1 runs
-src_ldap_port <portNo> : Port at which OID-1 runs
-src_ldap_dn <bindDn> : Bind Dn to connect to OID-1
-src_ldap_passwd <password> : Bind Dn password to connect to OID-1
-dst_ldap_host <hostName> : Host where OID-2 runs
-dst_ldap_port <portNo> : Port at which OID-2 runs
-dst_ldap_dn <bindDn> : Bind Dn to connect to OID-2
-dst_ldap_passwd <password> : Bind Dn password to connect to OID-2
-log <logFile> : Log file
```

Defaults:

```
src_ldap_host - localhost, src_ldap_port & dst_ldap_port - 389
src_ldap_dn & dst_ldap_dn - cn=orcladmin account
```

Examples:

```
dipassistant reassociate -src_ldap_host oid1.mycorp.com \
-dst_ldap_host oid2.mycorp.com -src_ldap_passwd xxxx \
-dst_ldap_passwd xxxx

dipassistant rs -help
```

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* (9.0.4)

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–22.

Table A–22 Limitations of Bootstrapping in the Directory Integration and Provisioning Assistant

Type of Bootstrapping	Limitation
LDIF-to-LDIF	None

Table A–22 (Cont.) Limitations of Bootstrapping in the Directory Integration and Provisioning Assistant

Type of Bootstrapping	Limitation
LDAP-to-LDIF	For a large number of entries, bootstrapping can fail with an error of size limit exceeded. To resolve this, the 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 the proprietary Import/Export tool, take the dump of the data, and 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

The ldapUploadAgentFile.sh Tool Syntax

Use `LdapUploadAgentFile.sh` to load mapping and configuration information when you are synchronizing directories.

```
ldapUploadAgentFile.sh -name  profile_name
-config configset_the_profile_is_associated_with
-LDAPhost  directory_server_host
-LDAPport  directory_server_port
-binddn  DN_that_can_modify_the_profile   >
-bindpass password_for_the_bind_DN
-attrtype  "MAP" | "ATTR"
-filename complete_path_of_file_to_be_uploaded
```

Table A–23 Arguments for ldapUploadAgentFile.sh

Argument	Description
`Name`	The name of the integration profile to which the information needs to be loaded.
`Config`	The configset to which the profile belongs to.
`LDAPhost`	Directory server host
`LDAPport`	Directory server port

Table A–23 (Cont.) Arguments for ldapUploadAgentFile.sh

Argument	Description
`Binddn`	Bind DN of the directory user who has access rights to modify the profile entry. `The default is cn=orcladmin`
`Bindpass`	Password corresponding to the bind DN. The default is `welcome`.
`AttrType`	Type of file to be loaded. "MAP' is specified for loading the mapping file. And "ATTR" is specified for loading the config info file.
`Filename`	Complete path name of the file to be uploaded.

Note: Alternatively, you can use the Directory Integration and Provisioning Assistant to perform this operation. Enter either of the following:

```
dipassistant mp [options] odip.profile.mapfile=your
map file


dipassistant mp [options] odip.profile.configfile=
your configuration file
```

See Also: Chapter 33, "Oracle Directory Synchronization Service" in *Oracle Internet Directory Administrator's Guide* for a description of when to use `ldapUploadAgentFile.sh`

The ldapCreateConn.sh Tool Syntax

You can create an integration profile by using the command-line tool ldapcreateConn.sh. This tool is in the following directory:

`$ORACLE_HOME/ldap/admin/.`

The following example creates an integration profile named "HRMS" in configuration set 2:

```
ldapcreateConn.sh
        -name agent_name>
        [ -type  <IMPORT | EXPORT > ] \
        [ -agentpwd agent_password ] \
        [ -config configset_to_associate_with ] \
```

```
[ -LDAPhost directory_server_host ]
[ -LDAPport directory_server_port ] \
[ -binddn DN_of_super_user] \
[ -bindpass   Bind_password ] \                                    [
[-retry   maximum_retry_count_on_synchronization_errors ] \
[ -poll polling_interval_for_synchronization ] \
[ -host host_on_which_to_run_agent ] \
[ -conndirurl  connected_directory_URL ] \
[ -conndiracct connected_directory_account_information ] \
[ -conndirpwd connected_directory_account_password ] \
[ -execmd command_line_for_the_agent ]  \
[ -iftype interface_type ]      \                                   [
-condirfilter connected_directory_matching_filter ]\
[ -oidfilter OID_matching_filter ] \
[ -U SSL_authentication_mode ]
[ -W wallet_location ]\
[ -P wallet_password ]
```

Table A–24 Arguments for Registering a Partner Agent by Using ldapcreateConn.sh

Argument	Description
Name	The name of the Integration Profile.This must be unique.
Type	IMPORT/EXPORT. The default is IMPORT/
Agentpwd	The password to protect the profile. The default is 'welcome'.
Config	The configuration set number. The default is 1.
LDAPhost	Directory server host. The default is the current host.
LDAPport	Directory server port The default is port 389.
Binddn	The bind DN of the Directory user which has the privileges to create Integration profile. The default is 'cn=orcladmin'
Bindpass	The bind password. The default is 'welcome'
Retry	Maximum number of retries to be done by the server when encountering a synchronization error. The default is '5'.
Poll	The scheduling interval of the profile. The default is '60' seconds.
Host	This is currently used. For the time being, it should be set to the machine name on which the DIP server is executing.
Conndirurl	The connected directory access Information.
Conndiracct	The connected directory account.

Table A–24 (Cont.) Arguments for Registering a Partner Agent by Using

Argument	Description
`Conndirpwd`	The connected directory account password
`Execmd`	The OS command line to execute the partner agent.
`Iftype`	The interface type. The default is TAGGED.
`Condirfilter`	The connected directory matching filter
`Oidfilter`	The OID matching filter.

Note: Alternatively, you can use the `createprofile` option of the Directory Integration and Provisioning Assistant to perform this operation.

The ldapDeleteConn.sh Tool Syntax

You can deregister a synchronization profile by using the command-line tool ldapDeleteConn`.sh`. This tool is in the directory `$`*`ORACLE_HOME`*`/ldap/admin/`.

The syntax is:

```
ldapdeleteConn.sh [ -name Profile_Name ]
 -LDAPhost <LDAP server host> (default is local host)]
            [ -LDAPport directory_server_port> (default 389)]
            [ -binddn SuperUserDN (default cn=orcladmin ) ]
            [ -bindpass   password (default=welcome) ]
            [ -config configset_associated_with_agent ]
            [ -U <SSL_authentication_mode> ]
            [ -W Wallet_location ]
            [ -P Wallet_password ]
            [ -help | -usage ]
```

The following example deregisters a profile entry and dissociates it from the configuration set 2 (`config 2`) entry:

```
ldapDeleteConn.sh name HRMS config 2
```

Note: Alternatively, you can use the `deleteprofile` option of the Directory Integration and Provisioning Assistant to perform this operation.

The StopOdiServer.sh Tool Syntax

In a client-only installation where OID Monitor and OIDCTL tools are not available, you can start the directory integration and provisioning server without OIDCTL. To stop the server, use the stopOdiServer.sh tool.

The path name for this tool is:
$*ORACLE_HOME*/ldap/admin/stopodiserver.sh

The usage is:

```
$ORACLE_HOME/ldap/admin/stopodiserver.sh
        [ -LDAPhost LDAP_server_host ]
        [ -LDAPport LDAP_server_port ]
        [ -binddn super_user_dn (default cn=orcladmin ) ]
        [ -bindpass   bind_password (default=welcome) ]
        -instance instance_number_to_stop
```

Table A–25 Arguments for Stopping the Oracle Directory Integration and Provisioning Server

Argument	Description
LDAPhost	Directory server host. The default is the current host.
LDAPport	Directory server port. The default is port 389.
Binddn	The bind DN of the Directory user which has the privileges to create Integration profile. The default is 'cn=orcladmin'
Bindpass	The bind password. The default is 'welcome'
Instance	The instance number of the Oracle directory integration and provisioning server to 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:
  http://sources.redhat.com
- MKS Toolkit 6.1. Visit:
  http://www.datafocus.com/

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_numbert
        -srcdn privileged_DN_in_source_directory_to_access_schema
        -srcpwd password
        -dsthost destination_LDAP_directory
        -dstport destination_LDAP_port
        -dstdn privileged_dn_in_destination_directory_to_access_schema
        -dstpwd password
        [-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–26 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–26.

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–26 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
`-lhost`	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

The Provisioning Subscription Tool (oidprovtool) Syntax

Use the Provisioning Subscription Tool to administer provisioning profile entries in the directory. More specifically, use it to perform these activities:

- Create a new provisioning profile. A new provisioning profile is created and set to the enabled state so that the Oracle Directory Integration and Provisioning platform can process it
- Disable an existing provisioning profile
- Enabled a disabled provisioning profile
- Delete an existing provisioning profile
- Get the current status of a given provisioning profile
- Clear all of the errors in an existing provisioning profile

The Provisioning Subscription Tool shields the location and schema details of the provisioning profile entries from the callers of the tool. From the callers' perspective, the combination of an application and a subscriber uniquely identify a provisioning profile. The constraint in the system is that there can be only one provisioning profile for each application for each subscriber.

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: `http://sources.redhat.com`
- MKS Toolkit 6.1. Visit: `http://www.datafocus.com/`

The name of the executable is oidProvTool, located in `$ORACLE_HOME/bin`.

To invoke this tool, use this command:

```
oidprovtool param1=param1_value  param2=param2_value param3=param3_value ...
```

The Provisioning Subscription Tool accepts the following parameters:

Table A–27 Provisioning Subscription Tool Parameters

Name	Description	Operations	Mandatory/Optional
`operation`	The subscription operation to be performed. The legal values for this parameter are: create, enable, disable, delete, status and reset. 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 subscription operations are to be performed. If not specified, the default value of 'localhost' is assumed.	all	O
profile_status	The status of the profile (`ENABLED`/ `DISABLED`). Default is `ENABLED`.	Create	O
profile_mode	`IBOUND`/`OUTBOUND`/`BOTH`. Default is `OUTBOUND`.	Create	O
profile_debug	The debugging level with which the profile is executed by the Oracle directory integration and provisioning server.	All	O
sslmode	Indicator of whether to execute the Provisioning Subscription Tool in SSL mode. A value of `0` indicates non-ssl and `1` indicates SSL mode.	All	O
`ldap_port`	The TCP/IP port on which the LDAP server is listening for requests. If not specified, the default value of '389' is assumed.	all	O

Table A–27 (Cont.) Provisioning Subscription Tool Parameters

Name	Description	Operations	Mandatory/Optional
`ldap_user_dn`	The LDAP distinguished name of the user on whose behalf the operation is to be performed. Not all users have the necessary permissions to perform Provisioning Subscription operations. Please see the administrative guide to grant or deny LDAP users the permission to perform Provisioning Subscription operations.	all	M
`ldap_user_password`	The password of the user on whose behalf the operation is to be performed.	all	M
`application_dn`	The LDAP distinguished name of the application for which the Provisioning Subscription Operation is being performed. The combination of the application_dn and the organization_dn parameters help the subscription tool to uniquely identify a provisioning profile.	all	M
`organization_dn`	The LDAP distinguished name of the organization for which the Provisioning Subscription Operation is being performed. The combination of the application_dn and the organization_dn parameters help the subscription tool to uniquely identify a provisioning profile.	all	M
`interface_name`	Database schema name for the PLSQL package. Format of the value should be: [Schema].[PACKAGE_NAME]	create only	M

Table A–27 (Cont.) Provisioning Subscription Tool Parameters

Name	Description	Operations	Mandatory/Optional
`interface_type`	The type of the interface to which events have to be propagated.Valid Values: PLSQL (if not specified this is assumed as the default)	create only	O
`interface_connect_info`	Database connect string Format of this string:[HOST]:[PORT]:[SID]: [USER_ID]:[PASSWORD]	create only	M
`interface_version`	The version of the interface protocol. Valid Values: 1.0 or 1.11.0 will be the old interface. If not specified, this is used as the default.	create only	O
`interface_additional_info`	Additional information for the interface. This is not currently used.	create only	O

Table A–27 (Cont.) Provisioning Subscription Tool Parameters

Name	Description	Operations	Mandatory/Optional
`schedule`	The scheduling information for this profile. The value is the length of the time interval in seconds after which DIP will process this profile. If not specified, a default of 3600 is assumed.	create only	O
`max_retries`	The number of times the Provisioning Service should retry a failed event delivery. If not specified, a default value of 5 is assumed.	create only	O
`event_subscription`	Events for which DIP should send notification to this application. Format of this string:"[USER]GROUP]:[*Domain of interest*>]:[DELETE]ADD]MODIFY(<*list of attributes separated by comma*>)]"Multiple values may be specified by listing the parameter multiple times each with different values. If not specified the following defaults are assumed:USER:<org. DN>:DELETEGROUP:<org. DN>:DELETEqQthat is, send user and group delete notifications under the organization DN.	create only	O

B

Sample Usage

This appendix provides sample code.

This section contains these topics

- DBMS_LDAP Sample Code
- DBMS_LDAP_UTL Sample Code
- Java Sample Code

DBMS_LDAP Sample Code

This section contains these topics:

Using DBMS_LDAP from a Database Trigger

The DBMS_LDAP API can be invoked from database triggers to synchronize any changes to a database table with an enterprise-wide LDAP server. The following example illustrates how changes to a table called 'EMP' are synchronized with the data in an LDAP server using triggers for insert, update, and delete. There are two files associated with this sample:

- The file `trigger.sql` creates the table as well as the triggers associated with it
- The file `empdata.sql` inserts some sample data into the table EMP, which automatically gets updated to the LDAP server through the insert trigger

These files can be found in the `plsql` directory under `$`*`ORACLE_HOME`*`/ldap/demo`

The trigger.sql File

This SQL file creates a database table called 'EMP' and creates a trigger on it called LDAP_EMP which will synchronize all changes happening to the table with an LDAP server. The changes to the database table are reflected/replicated to the LDAP directory using the DBMS_LDAP package.

This script assumes the following:

- LDAP server hostname: NULL (local host)
- LDAP server portnumber: 389
- Directory container for employee records: o=acme, dc=com
- Username/Password for Directory Updates: cn=orcladmin/welcome

The aforementioned variables could be customized for different environments by changing the appropriate variables in the code below.

Table Definition Employee Details(Columns) in Database Table(EMP):

EMP_ID—Number

FIRST_NAME—Varchar2

LAST_NAME—Varchar2

MANAGER_ID—Number

PHONE_NUMBER—Varchar2

MOBILE—Varchar2

ROOM_NUMBER—Varchar2

TITLE—Varchar2

LDAP Schema Definition & Mapping to Relational Schema EMP Corresponding Data representation in LDAP directory:

DN—cn=F*IRST_NAME LAST_NAME*, o=acme, dc=com]

cn—*FIRST_NAME LAST_NAME*

sn—*LAST_NAME*

givenname—*FIRST_NAME*

manager—DN

telephonenumber—*PHONE_NUMBER*

mobile—*MOBILE*

employeeNumber—*EMP_ID*

userpassword—FIRST_NAME

objectclass—person, organizationalperson, inetOrgPerson, top

```
—Creating EMP table

PROMPT Dropping Table EMP ..
drop table EMP;

PROMPT Creating Table EMP ..
CREATE TABLE EMP (
    EMP_ID    NUMBER,             Employee Number
    FIRST_NAME  VARCHAR2(256),    First Name
    LAST_NAME  VARCHAR2(256),     Last Name
    MANAGER_ID  NUMBER,           Manager Number
    PHONE_NUMBER VARCHAR2(256),   Telephone Number
    MOBILE   VARCHAR2(256),       Mobile Number
    ROOM_NUMBER  VARCHAR2(256),   Room Number
    TITLE   VARCHAR2(256)         Title in the company
);

—Creating Trigger LDAP_EMP

PROMPT Creating Trigger LDAP_EMP ..

CREATE OR REPLACE TRIGGER LDAP_EMP
AFTER INSERT OR DELETE OR UPDATE ON EMP
FOR EACH ROW

DECLARE
    retval   PLS_INTEGER;
    emp_session  DBMS_LDAP.session;
    emp_dn   VARCHAR2(256);
    emp_rdn   VARCHAR2(256);
    emp_array  DBMS_LDAP.MOD_ARRAY;
    emp_vals  DBMS_LDAP.STRING_COLLECTION ;
    ldap_host  VARCHAR2(256);
    ldap_port  VARCHAR2(256);
    ldap_user  VARCHAR2(256);
    ldap_passwd  VARCHAR2(256);
    ldap_base  VARCHAR2(256);
```

```
BEGIN

    retval       := -1;
    -- Customize the following variables as needed
    ldap_host  := NULL;
    ldap_port  := '389';
    ldap_user  := 'cn=orcladmin';
    ldap_passwd:= 'welcome';
    ldap_base  := 'o=acme,dc=com';
    -- end of customizable settings

    DBMS_OUTPUT.PUT('Trigger [LDAP_EMP]: Replicating changes ');
    DBMS_OUTPUT.PUT_LINE('to directory .. ');
    DBMS_OUTPUT.PUT_LINE(RPAD('LDAP Host ',25,' ') || ': ' || ldap_host);
    DBMS_OUTPUT.PUT_LINE(RPAD('LDAP Port ',25,' ') || ': ' || ldap_port);

    -- Choosing exceptions to be raised by DBMS_LDAP library.
    DBMS_LDAP.USE_EXCEPTION := TRUE;

    -- Initialize ldap library and get session handle.
    emp_session := DBMS_LDAP.init(ldap_host,ldap_port);

    DBMS_OUTPUT.PUT_LINE (RPAD('Ldap session ',25,' ')  || ': ' ||
        RAWTOHEX(SUBSTR(emp_session,1,8)) ||
        '(returned from init)');

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

        DBMS_OUTPUT.PUT_LINE(RPAD('simple_bind_s Returns ',25,' ') || ': '
              || TO_CHAR(retval));

    -- Process New Entry in the database

    IF INSERTING THEN

        -- Create and setup attribute array for the New entry
        emp_array := DBMS_LDAP.create_mod_array(14);

        -- RDN to be - cn="FIRST_NAME LAST_NAME"

        emp_vals(1) := :new.FIRST_NAME || ' ' || :new.LAST_NAME;

        DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_ADD,
```

```
'cn',emp_vals);

emp_vals(1) := :new.LAST_NAME;

DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_ADD,
            'sn',emp_vals);

emp_vals(1) := :new.FIRST_NAME;

DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_ADD,
            'givenname',emp_vals);

emp_vals(1) := 'top';
emp_vals(2) := 'person';
emp_vals(3) := 'organizationalPerson';
emp_vals(4) := 'inetOrgPerson';

DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_ADD,
            'objectclass',emp_vals);

emp_vals.DELETE;
emp_vals(1) := :new.PHONE_NUMBER;

DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_ADD,
            'telephonenumber',emp_vals);

emp_vals(1) := :new.MOBILE;

DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_ADD,
            'mobile',emp_vals);

emp_vals(1) := :new.ROOM_NUMBER;

DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_ADD,
            'roomNumber',emp_vals);

emp_vals(1) := :new.TITLE;

DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_ADD,
            'title',emp_vals);

emp_vals(1) := :new.EMP_ID;

DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_ADD,
            'employeeNumber',emp_vals);
```

```
    emp_vals(1) := :new.FIRST_NAME;

    DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_ADD,
                'userpassword',emp_vals);

    -- DN for Entry to be Added under 'ldap_base' [o=acme, dc=com]

    emp_dn := 'cn=' || :new.FIRST_NAME || ' ' ||
    :new.LAST_NAME || ', ' || ldap_base ;
    DBMS_OUTPUT.PUT_LINE(RPAD('Adding Entry for DN ',25,' ') || ': ['
               || emp_dn || ']');

    -- Add new Entry to ldap directory
    retval := DBMS_LDAP.add_s(emp_session,emp_dn,emp_array);
    DBMS_OUTPUT.PUT_LINE(RPAD('add_s Returns ',25,' ')  || ': '
               || TO_CHAR(retval));

    -- Free attribute array (emp_array)
    DBMS_LDAP.free_mod_array(emp_array);

END IF; -- INSERTING

-- Process Entry deletion in database

IF DELETING THEN

-- DN for Entry to be deleted under 'ldap_base' [o=acme, dc=com]

emp_dn := 'cn=' || :old.FIRST_NAME || ' ' ||
:old.LAST_NAME || ', ' || ldap_base ;
DBMS_OUTPUT.PUT_LINE(RPAD('Deleting Entry for DN ',25,' ') ||
         ': [' || emp_dn || ']');

-- Delete entry in ldap directory
retval := DBMS_LDAP.delete_s(emp_session,emp_dn);
     DBMS_OUTPUT.PUT_LINE(RPAD('delete_s Returns ',25,' ') || ': ' ||
           TO_CHAR(retval));

END IF; -- DELETING

-- Process updated Entry in database

IF UPDATING THEN
```

```
-- Since two Table columns(in this case) constitue a RDN
-- check for any changes and update RDN in ldap directory
-- before updating any other attributes of the Entry.

IF :old.FIRST_NAME <> :new.FIRST_NAME OR
     :old.LAST_NAME  <> :new.LAST_NAME THEN

    emp_dn := 'cn=' || :old.FIRST_NAME || ' ' ||
        :old.LAST_NAME || ', ' || ldap_base;

    emp_rdn := 'cn=' || :new.FIRST_NAME || ' ' || :new.LAST_NAME;

    DBMS_OUTPUT.PUT_LINE(RPAD('Renaming OLD DN ',25,' ') ||
              ': [' || emp_dn || ']');
    DBMS_OUTPUT.PUT_LINE(RPAD(' => NEW RDN ',25,' ') ||
              ': [' || emp_rdn || ']' );
    retval := DBMS_LDAP.modrdn2_s(emp_session,emp_dn,emp_rdn,
                DBMS_LDAP.MOD_DELETE);
    DBMS_OUTPUT.PUT_LINE(RPAD('modrdn2_s Returns ',25,' ') || ': ' ||
                TO_CHAR(retval));
END IF;

-- DN for Entry to be updated under 'ldap_base' [o=acme, dc=com]

emp_dn := 'cn=' || :new.FIRST_NAME || ' ' ||
   :new.LAST_NAME || ', ' || ldap_base;

DBMS_OUTPUT.PUT_LINE(RPAD('Updating Entry for DN ',25,' ') ||
            ': [' || emp_dn || ']');

-- Create and setup attribute array(emp_array) for updated entry
emp_array := DBMS_LDAP.create_mod_array(7);

emp_vals(1) := :new.LAST_NAME;

DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_REPLACE,
            'sn',emp_vals);

emp_vals(1) := :new.FIRST_NAME;

DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_REPLACE,
            'givenname',emp_vals);

emp_vals(1) := :new.PHONE_NUMBER;
```

```
    DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_REPLACE,
                'telephonenumber',emp_vals);

    emp_vals(1) := :new.MOBILE;

    DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_REPLACE,
                'mobile',emp_vals);

    emp_vals(1) := :new.ROOM_NUMBER;

    DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_REPLACE,
                'roomNumber',emp_vals);

    emp_vals(1) := :new.TITLE;

    DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_REPLACE,
                'title',emp_vals);

    emp_vals(1) := :new.EMP_ID;

    DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_REPLACE,
                'employeeNumber',emp_vals);

    -- Modify entry in ldap directory
      retval := DBMS_LDAP.modify_s(emp_session,emp_dn,emp_array);

       DBMS_OUTPUT.PUT_LINE(RPAD('modify_s Returns ',25,' ') || ': ' ||
                   TO_CHAR(retval));

    -- Free attribute array (emp_array)
    DBMS_LDAP.free_mod_array(emp_array);

END IF; -- UPDATING

-- Unbind from ldap directory
retval := DBMS_LDAP.unbind_s(emp_session);

DBMS_OUTPUT.PUT_LINE(RPAD('unbind_res Returns ',25,' ') ||  ': ' ||
          TO_CHAR(retval));

DBMS_OUTPUT.PUT_LINE('Directory operation Successful .. exiting');

-- Handle Exceptions
EXCEPTION
    WHEN OTHERS THEN
```

```
        -- TODO : should the trigger call unbind at this point ??
        --   what if the exception was raised from unbind itself ??

        DBMS_OUTPUT.PUT_LINE(' Error code    : ' || TO_CHAR(SQLCODE));
        DBMS_OUTPUT.PUT_LINE(' Error Message : ' || SQLERRM);
        DBMS_OUTPUT.PUT_LINE(' Exception encountered .. exiting');

END;
/
-------------------------------END OF trigger.sql--------------------------
```

Using DBMS_LDAP for a Search

The following example illustrates using the DBMS_LDAP API to perform an LDAP search in a PL/SQL program. This example searches for the entries created using the trigger example described previously. It assumes a base of `o=acme,dc=com` and performs a subtree search to retrieve all entries that are subordinates of the base entry. The code shown below is contained in a file called `search.sql` which can be found in the `$ORACLE_HOME/ldap/demo/plsql` directory.

The search.sql File

This SQL file contains the PL/SQL code required to perform a typical search against an LDAP server.

This script assumes the following:

- LDAP server host name: `NULL` (local host)
- LDAP server portnumber: `389`
- Directory container for employee records: `o=acme, dc=com`
- Username/Password for Directory Updates: `cn=orcladmin/welcome`

> **Note:** Run this file after you have run the `trigger.sql` and `empdata.sql` scripts to see what entries were added by the database triggers.

```
set serveroutput on size 30000

DECLARE
    retval       PLS_INTEGER;
```

```
    my_session   DBMS_LDAP.session;
    my_attrs     DBMS_LDAP.string_collection;
    my_message   DBMS_LDAP.message;
    my_entry     DBMS_LDAP.message;
    entry_index  PLS_INTEGER;
    my_dn        VARCHAR2(256);
    my_attr_name VARCHAR2(256);
    my_ber_elmt  DBMS_LDAP.ber_element;
    attr_index   PLS_INTEGER;
    i          PLS_INTEGER;
    my_vals DBMS_LDAP.STRING_COLLECTION ;
    ldap_host  VARCHAR2(256);
    ldap_port  VARCHAR2(256);
    ldap_user  VARCHAR2(256);
    ldap_passwd  VARCHAR2(256);
    ldap_base  VARCHAR2(256);

BEGIN
    retval         := -1;

    -- Please customize the following variables as needed
    ldap_host  := NULL ;
    ldap_port  := '389';
    ldap_user  := 'cn=orcladmin';
    ldap_passwd:= 'welcome';
    ldap_base  := 'o=acme,dc=com';
    -- end of customizable settings

     DBMS_OUTPUT.PUT('DBMS_LDAP Search Example ');
     DBMS_OUTPUT.PUT_LINE('to directory .. ');
     DBMS_OUTPUT.PUT_LINE(RPAD('LDAP Host ',25,' ') || ': ' || ldap_host);
     DBMS_OUTPUT.PUT_LINE(RPAD('LDAP Port ',25,' ') || ': ' || ldap_port);

     -- Choosing exceptions to be raised by DBMS_LDAP library.
     DBMS_LDAP.USE_EXCEPTION := TRUE;


     my_session := DBMS_LDAP.init(ldap_host,ldap_port);

     DBMS_OUTPUT.PUT_LINE (RPAD('Ldap session ',25,' ')  || ': ' ||
         RAWTOHEX(SUBSTR(my_session,1,8)) ||
         '(returned from init)');

     -- bind to the directory
     retval := DBMS_LDAP.simple_bind_s(my_session,
```

```
            ldap_user, ldap_passwd);

 DBMS_OUTPUT.PUT_LINE(RPAD('simple_bind_s Returns ',25,' ') || ': '
          || TO_CHAR(retval));

 -- issue the search
 my_attrs(1) := '*'; -- retrieve all attributes
 retval := DBMS_LDAP.search_s(my_session, ldap_base,
                              DBMS_LDAP.SCOPE_SUBTREE,
                              'objectclass=*',
                              my_attrs,
                              0,
                              my_message);

 DBMS_OUTPUT.PUT_LINE(RPAD('search_s Returns ',25,' ') || ': '
          || TO_CHAR(retval));
 DBMS_OUTPUT.PUT_LINE (RPAD('LDAP message  ',25,' ')  || ': ' ||
     RAWTOHEX(SUBSTR(my_message,1,8)) ||
     '(returned from search_s)');

 -- count the number of entries returned
 retval := DBMS_LDAP.count_entries(my_session, my_message);
 DBMS_OUTPUT.PUT_LINE(RPAD('Number of Entries ',25,' ') || ': '
          || TO_CHAR(retval));
 DBMS_OUTPUT.PUT_
LINE('-------------------------------------------------');


 -- get the first entry
 my_entry := DBMS_LDAP.first_entry(my_session, my_message);
 entry_index := 1;

 -- Loop through each of the entries one by one
 while my_entry IS NOT NULL loop
       -- print the current entry
       my_dn := DBMS_LDAP.get_dn(my_session, my_entry);
       -- DBMS_OUTPUT.PUT_LINE ('        entry #' || TO_CHAR(entry_index) ||
        -- ' entry ptr: ' || RAWTOHEX(SUBSTR(my_entry,1,8)));
       DBMS_OUTPUT.PUT_LINE ('        dn: ' || my_dn);
       my_attr_name := DBMS_LDAP.first_attribute(my_session,my_entry,
       my_ber_elmt);
       attr_index := 1;
       while my_attr_name IS NOT NULL loop
             my_vals := DBMS_LDAP.get_values (my_session, my_entry,
             my_attr_name);
```

```
                if my_vals.COUNT > 0 then
                  FOR i in my_vals.FIRST..my_vals.LAST loop
                    DBMS_OUTPUT.PUT_LINE('          ' || my_attr_name || ' : '
            ||
                    SUBSTR(my_vals(i),1,200));
                  end loop;
             end if;
             my_attr_name := DBMS_LDAP.next_attribute(my_session,my_entry,
             my_ber_elmt);
             attr_index := attr_index+1;
       end loop;
       my_entry := DBMS_LDAP.next_entry(my_session, my_entry);
       DBMS_OUTPUT.PUT_
    LINE('================================================');
       entry_index := entry_index+1;
 end loop;

 -- unbind from the directory
 retval := DBMS_LDAP.unbind_s(my_session);
 DBMS_OUTPUT.PUT_LINE(RPAD('unbind_res Returns ',25,' ') ||  ': ' ||
          TO_CHAR(retval));

 DBMS_OUTPUT.PUT_LINE('Directory operation Successful .. exiting');

-- Handle Exceptions
    EXCEPTION
        WHEN OTHERS THEN
              DBMS_OUTPUT.PUT_LINE(' Error code    : ' || TO_CHAR(SQLCODE));
              DBMS_OUTPUT.PUT_LINE(' Error Message : ' || SQLERRM);
              DBMS_OUTPUT.PUT_LINE(' Exception encountered .. exiting');
END;
/
```

DBMS_LDAP_UTL Sample Code

This section contains these topics:

Example: User-Related Functions

This is a sample usage of user-related functions in the DBMS_LDAP_UTL package. You can create a user handle using DN, GUID or a simple name representing the user.

This sample program demonstrates the following user-related functions:

- DBMS_LDAP_UTL.create_user_handle()
- DBMS_LDAP_UTL.set_user_handle_properties()
- DBMS_LDAP_UTL.authenticate_user()
- DBMS_LDAP_UTL.get_user_properties()
- DBMS_LDAP_UTL.set_user_properties()

```
set serveroutput on size 30000

DECLARE

 ldap_host      VARCHAR2(256);
 ldap_port      PLS_INTEGER;
 ldap_user      VARCHAR2(256);
 ldap_passwd    VARCHAR2(256);
 ldap_base      VARCHAR2(256);

 retval              PLS_INTEGER;
 my_session          DBMS_LDAP.session;

 subscriber_handle   DBMS_LDAP_UTL.HANDLE;
 sub_type            PLS_INTEGER;
 subscriber_id       VARCHAR2(2000);
```

```
 my_pset_coll        DBMS_LDAP_UTL.PROPERTY_SET_COLLECTION;
 my_property_names   DBMS_LDAP.STRING_COLLECTION;
 my_property_values  DBMS_LDAP.STRING_COLLECTION;

 user_handle         DBMS_LDAP_UTL.HANDLE;
 user_id             VARCHAR2(2000);
 user_type           PLS_INTEGER;
 user_password       VARCHAR2(2000);

 my_mod_pset         DBMS_LDAP_UTL.MOD_PROPERTY_SET;


 my_attrs            DBMS_LDAP.STRING_COLLECTION;

BEGIN


 -- Please customize the following variables as needed

 ldap_host     :=   NULL ;
 ldap_port     :=   389;
 ldap_user     :=   'cn=orcladmin';
 ldap_passwd   :=   'welcome';

 sub_type      :=   DBMS_LDAP_UTL.TYPE_DN;
 subscriber_id :=   'o=acme,dc=com';
 user_type     :=   DBMS_LDAP_UTL.TYPE_DN;
 user_id       :=   'cn=user1,cn=users,o=acme,dc=com';
 user_password :=   'welcome';


 -- Choosing exceptions to be raised by DBMS_LDAP library.
 DBMS_LDAP.USE_EXCEPTION := TRUE;

 ------------------------------------------------
 -- Connect to the LDAP server
 -- and obtain and ld session.
 ------------------------------------------------

 my_session := DBMS_LDAP.init(ldap_host,ldap_port);

 ------------------------------------------------
 -- Bind to the directory
 --
 ------------------------------------------------
```

```
 retval := DBMS_LDAP.simple_bind_s(my_session,
                                   ldap_user,
                                   ldap_passwd);

 ------------------------------------------------------------------------
 -- Create Subscriber Handle
 --
 ------------------------------------------------------------------------

 retval := DBMS_LDAP_UTL.create_subscriber_handle(subscriber_handle,
                                                  sub_type,
                                                  subscriber_id);

 IF retval != DBMS_LDAP_UTL.SUCCESS  THEN
    -- Handle Errors
    DBMS_OUTPUT.PUT_LINE('create_subscriber_handle returns : ' || to_
char(retval));
 END IF;

 ------------------------------------------------------------------------
 -- Create User Handle
 --
 ------------------------------------------------------------------------

 retval := DBMS_LDAP_UTL.create_user_handle(user_handle,user_type,user_id);

 IF retval != DBMS_LDAP_UTL.SUCCESS  THEN
    -- Handle Errors
    DBMS_OUTPUT.PUT_LINE('create_user_handle returns : ' || to_char(retval));
 END IF;

 ------------------------------------------------------------------------
 -- Set user handle properties
 -- (link subscriber to user )
 ------------------------------------------------------------------------

 retval := DBMS_LDAP_UTL.set_user_handle_properties(user_handle,
                                                    DBMS_LDAP_UTL.SUBSCRIBER_HANDLE,
                                                    subscriber_handle);

 IF retval != DBMS_LDAP_UTL.SUCCESS  THEN
    -- Handle Errors
    DBMS_OUTPUT.PUT_LINE('set_user_handle_properties returns : ' || to_
char(retval));
```

```
END IF;

-------------------------------------------------------------------------
-- Authenticate User
--
-------------------------------------------------------------------------

retval := DBMS_LDAP_UTL.authenticate_user(my_session,
                                          user_handle,
                                          DBMS_LDAP_UTL.AUTH_SIMPLE,
                                          user_password,
                                          NULL);

IF retval != DBMS_LDAP_UTL.SUCCESS  THEN
   -- Handle Errors
   DBMS_OUTPUT.PUT_LINE('authenticate_user returns : ' || to_char(retval));
END IF;

-------------------------------------------------------------------------
-- Retrieve User Properties
--
-------------------------------------------------------------------------
-- like .. telephone number

my_attrs(1) := 'telephonenumber';

retval := DBMS_LDAP_UTL.get_user_properties(my_session,
                                        user_handle,
                                        my_attrs,
                                        DBMS_LDAP_UTL.ENTRY_PROPERTIES,
                                        my_pset_coll);

IF retval != DBMS_LDAP_UTL.SUCCESS  THEN
   -- Handle Errors
   DBMS_OUTPUT.PUT_LINE('get_user_properties returns : ' || to_char(retval));
END IF;

-------------------------------------------------------------------------
-- Modifying User Properties
--
-------------------------------------------------------------------------


retval := DBMS_LDAP_UTL.create_mod_propertyset(DBMS_LDAP_UTL.ENTRY_PROPERTIES,
                                           NULL,my_mod_pset);
```

```
 IF retval != DBMS_LDAP_UTL.SUCCESS  THEN
    -- Handle Errors
    DBMS_OUTPUT.PUT_LINE('create_mod_propertyset returns : ' || to_
char(retval));
 END IF;


 my_property_values.delete;
 my_property_values(1) := '444-6789';
 retval := DBMS_LDAP_UTL.populate_mod_propertyset(my_mod_pset,
                                          DBMS_LDAP_UTL.REPLACE_PROPERTY,
                                          'telephonenumber',my_property_
values);
 my_property_values.delete;

 IF retval != DBMS_LDAP_UTL.SUCCESS  THEN
    -- Handle Errors
    DBMS_OUTPUT.PUT_LINE('populate_mod_propertyset returns : ' || to_
char(retval));
 END IF;


 retval := DBMS_LDAP_UTL.set_user_properties(my_session,user_handle,
                                          DBMS_LDAP_UTL.ENTRY_PROPERTIES,
                                          my_mod_pset,
                                          DBMS_LDAP_UTL.MODIFY_PROPERTY_SET);

 IF retval != DBMS_LDAP_UTL.SUCCESS  THEN
    -- Handle Errors
    DBMS_OUTPUT.PUT_LINE('set_user_properties returns : ' || to_char(retval));
 END IF;


 ------------------------------------------
 -- Free Mod Propertyset
 --
 ------------------------------------------

 DBMS_LDAP_UTL.free_mod_propertyset(my_mod_pset);


 ---------------------------------------------------------------------
 -- Free handles
 --
```

```
 ---------------------------------------------------------------------

 DBMS_LDAP_UTL.free_handle(subscriber_handle);
 DBMS_LDAP_UTL.free_handle(user_handle);


  -- unbind from the directory
 retval := DBMS_LDAP.unbind_s(my_session);

 IF retval != DBMS_LDAP_UTL.SUCCESS  THEN
    -- Handle Errors
    DBMS_OUTPUT.PUT_LINE('unbind_s returns : ' || to_char(retval));
 END IF;


-- Handle Exceptions
 EXCEPTION
  WHEN OTHERS THEN
   DBMS_OUTPUT.PUT_LINE(' Error code    : ' || TO_CHAR(SQLCODE));
   DBMS_OUTPUT.PUT_LINE(' Error Message : ' || SQLERRM);
   DBMS_OUTPUT.PUT_LINE(' Exception encountered .. exiting');

  END;
/
```

Example: Property-Related Subprograms

This sample code demonstrates the usage of the Property related subprograms of the DBMS_LDAP_UTL package. Most of the subprograms related to user, subscriber, and group handles return DBMS_LDAP_UTL.PROPERTY_SET_ COLLECTION.

A PROPERTY_SET_COLLECTION contains a set of PROPERTY_SETs. A PROPERTY_SET is analogous to an LDAP entry which is identified by the DN. Each PropertySet contains a set of zero or more Properties. A Property is analogous to a particular attribute of an LDAP entry and it may contain one or more values.

```
set serveroutput on size 30000

DECLARE

 ldap_host      VARCHAR2(256);
 ldap_port      PLS_INTEGER;
 ldap_user      VARCHAR2(256);
 ldap_passwd    VARCHAR2(256);
```

```
 ldap_base      VARCHAR2(256);

 retval              PLS_INTEGER;
 my_session          DBMS_LDAP.session;

 subscriber_handle   DBMS_LDAP_UTL.HANDLE;
 sub_type            PLS_INTEGER;
 subscriber_id       VARCHAR2(2000);

 my_pset_coll        DBMS_LDAP_UTL.PROPERTY_SET_COLLECTION;
 my_property_names   DBMS_LDAP.STRING_COLLECTION;
 my_property_values  DBMS_LDAP.STRING_COLLECTION;

 user_handle         DBMS_LDAP_UTL.HANDLE;
 user_id             VARCHAR2(2000);
 user_type           PLS_INTEGER;
 user_password       VARCHAR2(2000);

 my_mod_pset         DBMS_LDAP_UTL.MOD_PROPERTY_SET;


 my_attrs            DBMS_LDAP.STRING_COLLECTION;

BEGIN


 -- Please customize the following variables as needed

 ldap_host     := NULL ;
 ldap_port     := 389;
 ldap_user     := 'cn=orcladmin';
 ldap_passwd   := 'welcome';

 sub_type      := DBMS_LDAP_UTL.TYPE_DN;
 subscriber_id := 'o=acme,dc=com';
 user_type     := DBMS_LDAP_UTL.TYPE_DN;
 user_id       := 'cn=user1,cn=users,o=acme,dc=com';
 user_password := 'welcome';


 -- Choosing exceptions to be raised by DBMS_LDAP library.
 DBMS_LDAP.USE_EXCEPTION := TRUE;

 ------------------------------------------------
 -- Connect to the LDAP server
```

```
-- and obtain and ld session.
-----------------------------------------------

my_session := DBMS_LDAP.init(ldap_host,ldap_port);

-----------------------------------------------
-- Bind to the directory
--
-----------------------------------------------

retval := DBMS_LDAP.simple_bind_s(my_session,
                                  ldap_user,
                                  ldap_passwd);

-----------------------------------------------------------------------
-- Create Subscriber Handle
--
-----------------------------------------------------------------------

retval := DBMS_LDAP_UTL.create_subscriber_handle(subscriber_handle,
                                                 sub_type,
                                                 subscriber_id);

IF retval != DBMS_LDAP_UTL.SUCCESS  THEN
   -- Handle Errors
   DBMS_OUTPUT.PUT_LINE('create_subscriber_handle returns : ' || to_
char(retval));
END IF;

-----------------------------------------------------------------------
-- Create User Handle
--
-----------------------------------------------------------------------

retval := DBMS_LDAP_UTL.create_user_handle(user_handle,user_type,user_id);

IF retval != DBMS_LDAP_UTL.SUCCESS  THEN
   -- Handle Errors
   DBMS_OUTPUT.PUT_LINE('create_user_handle returns : ' || to_char(retval));
END IF;

-----------------------------------------------------------------------
-- Set user handle properties
-- (link subscriber to user )
-----------------------------------------------------------------------
```

```
 retval := DBMS_LDAP_UTL.set_user_handle_properties(user_handle,
                                          DBMS_LDAP_UTL.SUBSCRIBER_HANDLE,
                                          subscriber_handle);

 IF retval != DBMS_LDAP_UTL.SUCCESS  THEN
    -- Handle Errors
    DBMS_OUTPUT.PUT_LINE('set_user_handle_properties returns : ' || to_
char(retval));
 END IF;

 ----------------------------------------------------------------------
 -- Retrieve User Properties
 --
 ----------------------------------------------------------------------
 -- like .. telephone number

 my_attrs(1) := 'telephonenumber';

 retval := DBMS_LDAP_UTL.get_user_properties(my_session,
                                       user_handle,
                                       my_attrs,
                                       DBMS_LDAP_UTL.ENTRY_PROPERTIES,
                                       my_pset_coll);

 IF retval != DBMS_LDAP_UTL.SUCCESS  THEN
    -- Handle Errors
    DBMS_OUTPUT.PUT_LINE('get_user_properties returns : ' || to_char(retval));
 END IF;

 ----------------------------------------------------------------------
 -- Print properties obtained for the user.
 --
 ----------------------------------------------------------------------
  IF my_pset_coll.count > 0 THEN


      FOR i in my_pset_coll.first .. my_pset_coll.last LOOP

       retval := DBMS_LDAP_UTL.get_property_names(my_pset_coll(i),
                                                  my_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(my_pset_coll(i),
```

```
                                                        my_property_names(j),
                                                        my_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;
              END IF;

            END LOOP;

          END IF; -- IF my_property_names.count > 0

       END LOOP;

   END IF; -- If my_pset_coll.count > 0


    -- Free my_properties
    IF my_pset_coll.count > 0 then
      DBMS_LDAP_UTL.free_propertyset_collection(my_pset_coll);
    end if;

  ------------------------------------------------------------------------
  -- Free handles
  --
  ------------------------------------------------------------------------

  DBMS_LDAP_UTL.free_handle(subscriber_handle);
  DBMS_LDAP_UTL.free_handle(user_handle);


   -- unbind from the directory
  retval := DBMS_LDAP.unbind_s(my_session);

  IF retval != DBMS_LDAP_UTL.SUCCESS  THEN
     -- Handle Errors
     DBMS_OUTPUT.PUT_LINE('unbind_s returns : ' || to_char(retval));
  END IF;


 -- Handle Exceptions
  EXCEPTION
```

```
  WHEN OTHERS THEN
   DBMS_OUTPUT.PUT_LINE(' Error code    : ' || TO_CHAR(SQLCODE));
   DBMS_OUTPUT.PUT_LINE(' Error Message : ' || SQLERRM);
   DBMS_OUTPUT.PUT_LINE(' Exception encountered .. exiting');

  END;
/
```

Example: Subscriber-Related Functions

This is a sample usage of Subscriber related functions in the DBMS_LDAP_UTL package. You can create a subscriber handle using DN, GUID or a simple name representing the subscriber.

This sample program demonstrates the following subscriber-related functions:

- DBMS_LDAP_UTL.create_subscriber_handle()
- DBMS_LDAP_UTL.get_subscriber_properties()

```
set serveroutput on size 30000

DECLARE

 ldap_host      VARCHAR2(256);
 ldap_port      PLS_INTEGER;
 ldap_user      VARCHAR2(256);
 ldap_passwd    VARCHAR2(256);
 ldap_base      VARCHAR2(256);

 retval              PLS_INTEGER;
 my_session          DBMS_LDAP.session;

 subscriber_handle   DBMS_LDAP_UTL.HANDLE;
 sub_type            PLS_INTEGER;
 subscriber_id       VARCHAR2(2000);

 my_pset_coll        DBMS_LDAP_UTL.PROPERTY_SET_COLLECTION;
 my_property_names   DBMS_LDAP.STRING_COLLECTION;
 my_property_values  DBMS_LDAP.STRING_COLLECTION;

 user_handle         DBMS_LDAP_UTL.HANDLE;
 user_id             VARCHAR2(2000);
 user_type           PLS_INTEGER;
 user_password       VARCHAR2(2000);
```

```
 my_mod_pset         DBMS_LDAP_UTL.MOD_PROPERTY_SET;


 my_attrs            DBMS_LDAP.STRING_COLLECTION;

BEGIN


 -- Please customize the following variables as needed

 ldap_host     := NULL ;
 ldap_port     := 389;
 ldap_user     := 'cn=orcladmin';
 ldap_passwd   := 'welcome';

 sub_type      := DBMS_LDAP_UTL.TYPE_DN;
 subscriber_id := 'o=acme,dc=com';
 user_type     := DBMS_LDAP_UTL.TYPE_DN;
 user_id       := 'cn=user1,cn=users,o=acme,dc=com';
 user_password := 'welcome';


 -- Choosing exceptions to be raised by DBMS_LDAP library.
 DBMS_LDAP.USE_EXCEPTION := TRUE;

 ------------------------------------------------
 -- Connect to the LDAP server
 -- and obtain and ld session.
 ------------------------------------------------

 my_session := DBMS_LDAP.init(ldap_host,ldap_port);

 ------------------------------------------------
 -- Bind to the directory
 --
 ------------------------------------------------

 retval := DBMS_LDAP.simple_bind_s(my_session,
                                   ldap_user,
                                   ldap_passwd);

 ------------------------------------------------------------------------
 -- Create Subscriber Handle
 --
 ------------------------------------------------------------------------
```

```
 retval := DBMS_LDAP_UTL.create_subscriber_handle(subscriber_handle,
                                                  sub_type,
                                                  subscriber_id);

 IF retval != DBMS_LDAP_UTL.SUCCESS  THEN
    -- Handle Errors
    DBMS_OUTPUT.PUT_LINE('create_subscriber_handle returns : ' || to_
char(retval));
 END IF;

 ----------------------------------------------------------------------
 -- Retrieve Subscriber Properties
 --
 ----------------------------------------------------------------------
 -- like .. telephone number

 my_attrs(1) := 'orclguid';

 retval := DBMS_LDAP_UTL.get_subscriber_properties(my_session,
                                         subscriber_handle,
                                         my_attrs,
                                         DBMS_LDAP_UTL.ENTRY_PROPERTIES,
                                         my_pset_coll);

 IF retval != DBMS_LDAP_UTL.SUCCESS  THEN
    -- Handle Errors
    DBMS_OUTPUT.PUT_LINE('get_subscriber_properties returns : ' || to_
char(retval));
 END IF;

 ----------------------------------------------------------------------
 -- Free handle
 --
 ----------------------------------------------------------------------

 DBMS_LDAP_UTL.free_handle(subscriber_handle);


  -- unbind from the directory
 retval := DBMS_LDAP.unbind_s(my_session);

 IF retval != DBMS_LDAP_UTL.SUCCESS  THEN
    -- Handle Errors
    DBMS_OUTPUT.PUT_LINE('unbind_s returns : ' || to_char(retval));
```

```
 END IF;


-- Handle Exceptions
 EXCEPTION
  WHEN OTHERS THEN
   DBMS_OUTPUT.PUT_LINE(' Error code    : ' || TO_CHAR(SQLCODE));
   DBMS_OUTPUT.PUT_LINE(' Error Message : ' || SQLERRM);
   DBMS_OUTPUT.PUT_LINE(' Exception encountered .. exiting');

  END;
/
```

Example: Group-Related Functions

This is a sample usage of Group related functions in DBMS_LDAP_UTL package. You can create a group handle using DN, GUID or a simple name representing the group.

This sample program demonstrates the following group-related functions:

- DBMS_LDAP_UTL.create_group_handle()
- DBMS_LDAP_UTL.set_group_handle_properties()
- DBMS_LDAP_UTL.check_group_membership()
- DBMS_LDAP_UTL.get_group_membership()
- DBMS_LDAP_UTL.get_group_properties()

```
set serveroutput on size 30000

DECLARE

 ldap_host      VARCHAR2(256);
 ldap_port      PLS_INTEGER;
 ldap_user      VARCHAR2(256);
 ldap_passwd    VARCHAR2(256);
 ldap_base      VARCHAR2(256);

 retval              PLS_INTEGER;
 my_session          DBMS_LDAP.session;

 subscriber_handle   DBMS_LDAP_UTL.HANDLE;
 sub_type            PLS_INTEGER;
 subscriber_id       VARCHAR2(2000);
```

```
 my_pset_coll        DBMS_LDAP_UTL.PROPERTY_SET_COLLECTION;
 my_property_names   DBMS_LDAP.STRING_COLLECTION;
 my_property_values  DBMS_LDAP.STRING_COLLECTION;

 group_handle         DBMS_LDAP_UTL.HANDLE;
 group_id             VARCHAR2(2000);
 group_type           PLS_INTEGER;

 user_handle         DBMS_LDAP_UTL.HANDLE;
 user_id             VARCHAR2(2000);
 user_type           PLS_INTEGER;

 my_mod_pset         DBMS_LDAP_UTL.MOD_PROPERTY_SET;


 my_attrs            DBMS_LDAP.STRING_COLLECTION;

BEGIN


 -- Please customize the following variables as needed

 ldap_host     := NULL ;
 ldap_port     := 389;
 ldap_user     := 'cn=orcladmin';
 ldap_passwd   := 'welcome';

 sub_type      := DBMS_LDAP_UTL.TYPE_DN;
 subscriber_id := 'o=acme,dc=com';
 user_type     := DBMS_LDAP_UTL.TYPE_DN;
 user_id       := 'cn=user1,cn=users,o=acme,dc=com';
 group_type     := DBMS_LDAP_UTL.TYPE_DN;
 group_id       := 'cn=group1,cn=groups,o=acme,dc=com';


 -- Choosing exceptions to be raised by DBMS_LDAP library.
 DBMS_LDAP.USE_EXCEPTION := TRUE;

 ------------------------------------------------
 -- Connect to the LDAP server
 -- and obtain and ld session.
 ------------------------------------------------

 my_session := DBMS_LDAP.init(ldap_host,ldap_port);
```

```
-----------------------------------------------
-- Bind to the directory
--
-----------------------------------------------

retval := DBMS_LDAP.simple_bind_s(my_session,
                                  ldap_user,
                                  ldap_passwd);

-----------------------------------------------------------------------
-- Create Subscriber Handle
--
-----------------------------------------------------------------------

retval := DBMS_LDAP_UTL.create_subscriber_handle(subscriber_handle,
                                                 sub_type,
                                                 subscriber_id);

IF retval != DBMS_LDAP_UTL.SUCCESS  THEN
   -- Handle Errors
   DBMS_OUTPUT.PUT_LINE('create_subscriber_handle returns : ' || to_
char(retval));
END IF;

-----------------------------------------------------------------------
-- Create User Handle
--
-----------------------------------------------------------------------

retval := DBMS_LDAP_UTL.create_user_handle(user_handle,user_type,user_id);

IF retval != DBMS_LDAP_UTL.SUCCESS  THEN
   -- Handle Errors
   DBMS_OUTPUT.PUT_LINE('create_user_handle returns : ' || to_char(retval));
END IF;

-----------------------------------------------------------------------
-- Set User handle properties
-- (link subscriber to user )
-----------------------------------------------------------------------

retval := DBMS_LDAP_UTL.set_user_handle_properties(user_handle,
                                                   DBMS_LDAP_UTL.SUBSCRIBER_HANDLE,
                                                   subscriber_handle);
```

```
 IF retval != DBMS_LDAP_UTL.SUCCESS  THEN
    -- Handle Errors
    DBMS_OUTPUT.PUT_LINE('set_user_handle_properties returns : ' || to_
char(retval));
 END IF;

 ------------------------------------------------------------------------
 -- Create Group Handle
 --
 ------------------------------------------------------------------------

 retval := DBMS_LDAP_UTL.create_group_handle(group_handle,group_type,group_id);

 IF retval != DBMS_LDAP_UTL.SUCCESS  THEN
    -- Handle Errors
    DBMS_OUTPUT.PUT_LINE('create_group_handle returns : ' || to_char(retval));
 END IF;

 ------------------------------------------------------------------------
 -- Set Group handle properties
 -- (link subscriber to group )
 ------------------------------------------------------------------------

 retval := DBMS_LDAP_UTL.set_group_handle_properties(group_handle,
                                        DBMS_LDAP_UTL.SUBSCRIBER_HANDLE,
                                        subscriber_handle);

 IF retval != DBMS_LDAP_UTL.SUCCESS  THEN
    -- Handle Errors
    DBMS_OUTPUT.PUT_LINE('set_group_handle_properties returns : ' || to_
char(retval));
 END IF;

 ------------------------------------------------------------------------
 -- Retrieve Group Properties
 --
 ------------------------------------------------------------------------
 -- like .. telephone number

 my_attrs(1) := 'uniquemember';

 retval := DBMS_LDAP_UTL.get_group_properties(my_session,
                                      group_handle,
                                      my_attrs,
```

```
                                        DBMS_LDAP_UTL.ENTRY_PROPERTIES,
                                        my_pset_coll);

 IF retval != DBMS_LDAP_UTL.SUCCESS  THEN
    -- Handle Errors
    DBMS_OUTPUT.PUT_LINE('get_group_properties returns : ' || to_char(retval));
 END IF;

 ---------------------------------------
 -- Check Group Membership
 --
 ---------------------------------------

 retval :=  DBMS_LDAP_UTL.check_group_membership( my_session,
                                  user_handle,
                                  group_handle,
                                  DBMS_LDAP_UTL.DIRECT_MEMBERSHIP);

 IF retval != DBMS_LDAP_UTL.SUCCESS  THEN
    -- Handle Errors
    DBMS_OUTPUT.PUT_LINE('check_group_membership returns : ' || to_
char(retval));
 END IF;

 ---------------------------------------
 -- Get Group Membership
 --
 ---------------------------------------

 my_attrs.delete();
 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 retval != DBMS_LDAP_UTL.SUCCESS  THEN
    -- Handle Errors
    DBMS_OUTPUT.PUT_LINE('get_group_membership returns : ' || to_char(retval));
 END IF;


 ------------------------------------------------------------------------
```

```
 -- Free handle
 --
 ------------------------------------------------------------------------

 DBMS_LDAP_UTL.free_handle(subscriber_handle);
 DBMS_LDAP_UTL.free_handle(user_handle);
 DBMS_LDAP_UTL.free_handle(group_handle);


  -- unbind from the directory
 retval := DBMS_LDAP.unbind_s(my_session);

 IF retval != DBMS_LDAP_UTL.SUCCESS  THEN
    -- Handle Errors
    DBMS_OUTPUT.PUT_LINE('unbind_s returns : ' || to_char(retval));
 END IF;


-- Handle Exceptions
 EXCEPTION
  WHEN OTHERS THEN
   DBMS_OUTPUT.PUT_LINE(' Error code    : ' || TO_CHAR(SQLCODE));
   DBMS_OUTPUT.PUT_LINE(' Error Message : ' || SQLERRM);
   DBMS_OUTPUT.PUT_LINE(' Exception encountered .. exiting');

  END;
/
```

Java Sample Code

This section contains Java sample code.

This section contains these topics:

User Class Sample Code

```
/*
 * SampleUser.java
 *
 * This is a sample usage of the User class in oracle.ldap.util package
 * found in ldapjclnt9.jar.  You can define a user using DN, GUID, or
 * a simple name representing the user.  The following methods are exercised
 * in this sample program:
 *
 * - User.authenticateUser() - to authenticate a user with the appropriate
 *   credentials
 * - User.getProperties() - to obtain properties of the user
 * - User.setProperties() - to add, replace, or delete properties of the user
 *
 */

import oracle.ldap.util.*;
import oracle.ldap.util.jndi.*;

import java.io.*;
import java.util.*;
import javax.naming.*;
import javax.naming.directory.*;

public class SampleUser {

   public static void main(String argv[])
          throws NamingException {
```

```
      // Create InitialDirContext

      InitialDirContext ctx = ConnectionUtil.getDefaultDirCtx( "sandal",
                                        "3060",
                                        "cn=orcladmin",
                                        "welcome" );

      // Create Subscriber object

      Subscriber mysub = null;

      try {
         // Creation using DN
         mysub = new Subscriber( ctx, Util.IDTYPE_DN, "o=oracle,dc=com", false
);
      }
      catch (UtilException e) {
         /*
          * Exception encountered in subscriber object constructor
          */
      }

      // Create User Objects

      User myuser = null,
           myuser1 = null;

      try {
         // Create User using a subscriber DN and the User DN

         myuser = new User ( ctx,
                             Util.IDTYPE_DN,
                             "cn=user1,cn=users,o=oracle,dc=com",
                             Util.IDTYPE_DN,
                             "o=oracle,dc=com",
                             false );

         // Create User using a subscriber object and the User
         // simple name

         myuser1 = new User ( ctx,
                              Util.IDTYPE_SIMPLE,
                              "user1",
                              mysub,
```

```
                        false );
}
catch ( UtilException e ) {
   /*
    * Exception encountered in User object constructor
    */
}

// Authenticate User
try {
   myuser1.authenticateUser(ctx,User.CREDTYPE_PASSWD,"welcome");
}
catch ( UtilException e ) {
   /*
    * Authenticate fails
    */
}

// Perform User operations

try {
   PropertySetCollection result = null;

   // Get telephonenumber of user

   String[] userAttrList = {"telephonenumber"};
   result = myuser1.getProperties(ctx,userAttrList);

   /*
    * Do work with result
          .
          .
          .
    */
    Util.printResults(result);

   // Set telephonenumber of user

   // Create JNDI ModificationItem

   ModificationItem[] mods = new ModificationItem[1];
   mods[0] = new ModificationItem(DirContext.REPLACE_ATTRIBUTE,
                    new BasicAttribute("telephonenumber", "444-6789"));

   // Perform modification using User object
```

```
            myuser.setProperties(ctx, mods);
         }
         catch ( UtilException e ) {
            /*
             * Exception encountered in User object operations
             */
         }
      }
   } // End of SampleUser.java
```

Subscriber Class Sample Code

```
/*
 * SampleSubscriber.java
 *
 * This is a sample usage of the Subscriber class in oracle.ldap.util package
 * found in ldapjclnt9.jar.  You can define a group using a DN, GUID, or a
 * simple name of the subscriber.  The following methods are exercised in
 * this sample program:
 *
 * - Subscriber.getProperties() - to obtain properties of the group
 *
 */

import oracle.ldap.util.*;
import oracle.ldap.util.jndi.*;

import java.io.*;
import java.util.*;
import javax.naming.*;
import javax.naming.directory.*;

public class SampleSubscriber {

   public static void main(String argv[])
          throws NamingException {

      // Create InitialDirContext

      InitialDirContext ctx = ConnectionUtil.getDefaultDirCtx( "sandal",
                                       "3060",
                                       "cn=orcladmin",
                                       "welcome" );
```

```
// Create Subscriber object

Subscriber mysub = null,
           mysub1 = null,
           mysub2 = null;
try {

   // Creation using DN
   mysub = new Subscriber( ctx,
                                    Util.IDTYPE_DN,
                                    "o=oracle,dc=com",
                                    false );

   // Creation using Simple Name
   mysub1 = new Subscriber( ctx,
                                     Util.IDTYPE_SIMPLE,
                                     "Oracle",
                                     false );

   // Creation using GUID
   mysub2 = new Subscriber( ctx,
                                     Util.IDTYPE_GUID,
                                     "93B37BBC3B1F46F8E034080020F73460",
                                     false );
}
catch (UtilException e) {
   /*
    * Exception encountered in subscriber object constructor
    */
}


// Set the attribute list for attributes returned
String[] attrList = { "cn",
                      "orclcommonusersearchbase",
                      "orclguid" };

// Get Subscriber Properties

PropertySetCollection result = null;
try {
   result = mysub.getProperties(ctx,attrList);
}
catch (UtilException e) {
   /*
```

```
            * Exception encountered when searching for subscriber properties
            */
         }

         /*
          * Do work with the result
          */

         Util.printResults(result);
      }
   }
```

Group Class Sample Code

```
/*
 * SampleGroup.java
 *
 * This is a sample usage of the Group class in oracle.ldap.util package
 * found in ldapjclnt9.jar.  You can define a group using DN or GUID.
 * The following methods are exercised in this sample program:
 *
 * - Group.isMember() - to see if a particular user is
 *   a member of this group
 * - Util.getGroupMembership() - to obtain the list of groups which a
 *   particular user belongs to
 * - Group.getProperties() - to obtain properties of the group
 *
 */

import oracle.ldap.util.*;
import oracle.ldap.util.jndi.*;

import java.io.*;
import java.util.*;
import javax.naming.*;
import javax.naming.directory.*;

public class SampleGroup {

   public static void main(String argv[])
          throws NamingException {

      // Create InitialDirContext

      InitialDirContext ctx = ConnectionUtil.getDefaultDirCtx( "sandal",
```

```
                                    "3060",
                                    "cn=orcladmin",
                                    "welcome" );

// Create Group Object
Group mygroup = null;
try {
   mygroup = new Group ( Util.IDTYPE_DN,
                         "cn=group1,cn=Groups,o=oracle,dc=com" );
}
catch ( UtilException e ) {
   /*
    * Error encountered in Group constructor
    */
}

// Create User Object

User myuser = null;
try {
   // Create User using a subscriber DN and the User DN
   myuser = new User ( ctx,
                       Util.IDTYPE_DN,
                       "cn=orcladmin,cn=users,o=oracle,dc=com",
                       Util.IDTYPE_DN,
                       "o=oracle,dc=com",
                       false );
}
catch ( UtilException e ) {
   /*
    * Exception encountered in User object constructor
    */
}

// Perform Group Operations

try {

   // isMember method

   if (mygroup.isMember( ctx,
                         myuser,
                         true ) ) {
      /*
       * myuser is a member of this group
```

```
                * Do work
                *           .
                *           .
                *           .
                */
                System.out.println("is member");
            }

            // Get all nested groups that a user belongs to

            PropertySetCollection result = Util.getGroupMembership( ctx,
                                                                     myuser,
                                                                     new String[0],
                                                                     true );
            /*
             * Do work with result
             *           .
             *           .
             *           .
             */
            Util.printResults ( result );

            // Get Group Properties

            result = getProperties( ctx, null );

            /*
             * Do work with result
             *           .
             *           .
             *           .
             */
        }
        catch ( UtilException e ) {
            /*
             * Exception encountered in getGroupMembership
             */
        }
    }
} // End of SampleGroup.java
```

Print Sample Code

```
/*
 * SamplePrint.java
```

```
 *
 * This sample program demonstrates the usage of the PropertySetCollection
 * class which is a key structure used in the oracle.ldap.util package for
 * obtaining search results.  A sample printResults() method is implemented
 * that neatly prints out the values of a PropertySetCollection.
 * A ProperSetCollection contains a set of PropertySets.  A PropertySet is
 * analogous to an LDAP entry which is identified by the DN.  Each PropertySet
 * contains a set of zero or more Properties.  A Property is analogous to a
 * particular attribute of an LDAP entry and it may contain one or more
 * values.  The printResults() method takes in a PropertySetCollection and
 * navigates through it in a systemmatic way, printing out the results to
 * the system output.
 *
 */

import oracle.ldap.util.*;
import oracle.ldap.util.jndi.*;

import java.io.*;
import java.util.*;
import javax.naming.*;
import javax.naming.directory.*;

public class SamplePrint {

   public static void printResults( PropertySetCollection resultSet )
   {
      // for loop to go through each PropertySet
      for (int i = 0; i < resultSet.size(); i++ )
      {
         // Get PropertySet
         PropertySet curEntry = resultSet.getPropertySet( i );
         Object obj = null;

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

         // Go through each Property of the PropertySet
         for (int j = 0; j < curEntry.size(); j++)
         {
            // Get Property
            Property curAttr = curEntry.getProperty( j );

            // Go through each value of the Property
            for (int k = 0; k < curAttr.size(); k++)
```

```
            {
              obj = curAttr.getValue(k);
              if( obj instanceof java.lang.String) {
               System.out.println( curAttr.getName() + ": "
                                + (String) obj);
              }
              else if (obj instanceof byte[]) {
               System.out.println( curAttr.getName() + ": "
                                + (new java.lang.String((byte [])obj)));
              }
            }
          }
          System.out.println();
        }
      }

} // End of SamplePrint.java
```

JNDI Sample Code

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

/*
* JNDI SASL Digest MD5 is available in JDK 1.4 and later
*/
public class LdapSaslDigestMD5
{
public static void main( String[] args)
throws Exception
{


System.out.println("port : " + args[1]);
System.out.println("bindDN : " + args[2]);
System.out.println("bindPwd: " + args[3]);

// Important note:
// The bindDN must be normalized before passing it to JNDI context
```

```
// For example: cn=smith,ou=oid,o=oracle,c=us
// (capital and space will not be accepted as a normalized dn)
// Right now we only support dn in only.
// uid form will be supported in the next release.

// The noralize dn call is a static method in Util.java.

String normDN = Util.normalizeDN(args[2]);

Hashtable hashtable = new Hashtable();

// Look through System Properties for Context Factory if available
// set the CONTEXT factory only if it has not been set
// in the environment - set default to com.sun.jndi.ldap.LdapCtxFactory
hashtable.put(Context.INITIAL_CONTEXT_FACTORY,
"com.sun.jndi.ldap.LdapCtxFactory");
hashtable.put(Context.PROVIDER_URL, "ldap://"+args[0]+":"+args[1]);
// Set security authentication context to Digest MD5
hashtable.put(Context.SECURITY_AUTHENTICATION, "DIGEST-MD5");
hashtable.put(Context.SECURITY_PRINCIPAL, normDN );
hashtable.put(Context.SECURITY_CREDENTIALS, args[3] );
hashtable.put("java.naming.security.sasl.realm", "");
LdapContext ctx = new InitialLdapContext(hashtable,null);
System.out.println("sasl bind successful");

// Some search after the SASL bind has been done
PropertySetCollection psc = Util.ldapSearch(ctx,"","objectclass=*",
SearchControls.OBJECT_SCOPE,
new String[] {"supportedSASLmechanism"});

Util.printResults(psc);

System.exit(0);

}
}


/*
* Sample code Using JNDI/SASL EXTERNAL to connect to OID
* This code will work only with OID SSL setup in mutual authentication mode
only.
* JNDI client needs to provide a client certificate that can be recognized by
* server side.
*/
```

```
import java.util.*;
import javax.naming.*;
import javax.naming.directory.*;
import oracle.security.jazn.spi.ldap.*;

public class LdapSaslExternal
{
public static void main (String[] args)
{
try {

Hashtable env = new Hashtable();

// Specify host and port to use for directory service
env.put("javax.net.debug", "all");
env.put("com.sun.jndi.ldap.trace.ber", System.out);
env.put("com.sun.naming.ldap.trace.ber", System.out);
env.put(Context.PROVIDER_URL, "ldap://some_url:5055/");

env.put("java.naming.security.protocol", "ssl");

System.setProperty("oracle.security.jazn.ldap.walletloc","<wallet_
url>/ewallet.txt");

System.setProperty("oracle.security.jazn.ldap.walletpwd","welcome01");

// You can use any SSL Socket Factory of your implementation or toolkit

env.put("java.naming.ldap.factory.socket","oracle.security.jazn.spi.ldap.JAZNSSL
SocketFactoryImpl");

// specify authentication information
// Note: you can also set security authentication context to "SIMPLE" to
// connect to OID; however, this functionality supports for backward
// compatibility with LDAP version 2.
env.put(Context.SECURITY_AUTHENTICATION, "EXTERNAL"); // TO-DO: add secure
hannes
env.put(Context.SECURITY_PRINCIPAL, "cn=test,ou=security,o=oracle,c=us");
nv.put(Context.SECURITY_CREDENTIALS, "welcome"); // TO-DO: add SSL

env.put("java.naming.factory.initial", "com.sun.jndi.ldap.LdapCtxFactory");

// Set your own SSL Socket factory Impl class here.
System.getProperties().put("SSLSocketFactoryImplClass","oracle.security.jazn.spi
```

```
.ldap.JAZNSSLSocketFactoryImpl");

DirContext dirCtx = new InitialDirContext(env);
System.out.println("return from InitialDirContext");
Object obj = dirCtx.lookup("");
System.out.println("Looked up obj : " + obj);
} catch (Exception exp) {
exp.printStackTrace();
System.exit(-1);
}
}
}
```

SASL-Based Authentication Sample Code

```
/* $Header: LdapSasl.java 05-may-2003.15:14:22 qdinh Exp $ */

/* Copyright (c) 2003, Oracle Corporation.  All rights reserved.  */

/*
   DESCRIPTION
    <short description of component this file declares/defines>

   PRIVATE CLASSES
    <list of private classes defined - with one-line descriptions>

   NOTES
    <other useful comments, qualifications, etc.>

   MODIFIED    (MM/DD/YY)
   *****       04/23/03 - Creation
 */

/**
 *  @version $Header: LdapSasl.java 05-may-2003.15:14:22 ***** Exp $
 *  @author  *****
 *  @since   release specific (what release of product did this appear in)
 */

package oracle.ldap.util.jndi;

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

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

public class LdapSasl
{
    public static void main( String[] args)
        throws Exception
    {


     System.out.println("port   : " + args[1]);
     System.out.println("bindDN : " + args[2]);
     System.out.println("bindPwd: " + args[3]);

     Hashtable hashtable = new Hashtable();

     // Look through System Properties for Context Factory if available
     // set the CONTEXT factory only if it has not been set
     // in the environment - set default to com.sun.jndi.ldap.LdapCtxFactory
     hashtable.put(Context.INITIAL_CONTEXT_FACTORY,
   "com.sun.jndi.ldap.LdapCtxFactory");

     hashtable.put(Context.PROVIDER_URL, "ldap://"+args[0]+":"+args[1]);

     //hashtable.put(Context.SECURITY_AUTHENTICATION, "simple");
     hashtable.put(Context.SECURITY_AUTHENTICATION, "DIGEST-MD5");
     hashtable.put(Context.SECURITY_PRINCIPAL, args[2] );
     hashtable.put(Context.SECURITY_CREDENTIALS, args[3] );
     hashtable.put("java.naming.security.sasl.realm", "");
     LdapContext ctx = new InitialLdapContext(hashtable,null);
     System.out.println("sasl bind successful");
     //PropertySetCollection psc =
Util.ldapSearch(ctx,"","objectclass=*",SearchControls.OBJECT_SCOPE,
     //new String[] {"supportedSASLmechanism"});

     //Util.printResults(psc);

     System.exit(0);

  }
}
```

C

DSML Syntax

This appendix contains the following sections:

- Capabilities of DSML
- DSML Syntax
- Tools Enabled for DSML

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 Using 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 universally 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 `namespace` URI [9] of DSMLv1 is `http://www.dsml.org/DSML`. All the XML element tags may be prefixed with `dsml` string. (that is, a `namespace` prefix).

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 is thus:

```
<dsml:dsml xmlns:dsml=http://www.dsml.org/DSML>
<!- 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`.

XML entry to describe the directory entry is as follows

```
<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:value>Svenugop@Oracle.com</dsml:value></dsml:attr>
  <dsml:attr name="givenname">
      <dsml:value>Siva V. Kumar</dsml:value></dsml:attr>
  <dsml:attr name="cn">
      <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`is 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>
```

In a similar way the directory attributes are also described. For example the attribute definition for the `cn` attribute may look like the following:

```
<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 Management Command-Line Tools" in Appendix A for complete syntax and usage information for these tools

The Oracle Internet Directory client tools `ldifwrite` generates directory data and schema LDIF files. If these LDIF files are converted to XML, then the XML file can be stored on an application server and queried. The response time to the client will be much less in this scenario compared to performing an LDAP operation as 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 Oracle9*i* 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 a cold failover cluster configuration, 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 a cold failover cluster configuration, 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 Oracle9*i*.

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*

Oracle9*i* Advanced Replication

A feature in Oracle9*i* 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 a cold failover cluster configuration, the cluster node on which the application runs at any given time.

See Also: **secondary node** on page Glossary-21

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 Oracle9*i* 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 a cold failover cluster configuration, the cluster node to which an application is moved during a failover.

See Also: **primary node** on page Glossary-18

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 a cold failover cluster configuration, the host name corresponding to this virtual IP address.

virtual IP address

In a cold failover cluster configuration, 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