

Oracle® Internet Directory

Application Developer's Guide

Release 9.2

March 2002

Part No. A96577-01

Part No. A96577-01

Copyright © 1999, 2002 Oracle Corporation. All rights reserved.

Priamry Author: Richard Smith

Contributing Authors: Torrance Brooksfuller, Sheryl Edwards

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

Graphic Artist: Valarie Moore

The Programs (which include both the software and documentation) contain proprietary information of Oracle Corporation; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent and other intellectual and industrial property laws. Reverse engineering, disassembly or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Oracle Corporation.

If the Programs are delivered to the U.S. Government or anyone licensing or using the programs on behalf of the U.S. Government, the following notice is applicable:

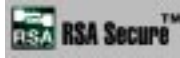
Restricted Rights Notice Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the Programs, including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication, and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-19, Commercial Computer Software - Restricted Rights (June, 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and Oracle Corporation disclaims liability for any damages caused by such use of the Programs.

Oracle is a registered trademark, and Oracle Names, Oracle Store, Oracle9i, PL/SQL, and SQL*Plus are trademarks or registered trademarks of Oracle Corporation. Other names may be trademarks of their respective owners.

Portions of this document are from "The C LDAP Application Program Interface," an Internet Draft of the Internet Engineering Task Force (Copyright (C) The Internet Society (1997-1999). All Rights Reserved), which expires on 8 April 2000. These portions are used in accordance with the following IETF directives: "This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing

Internet standards in which case the procedures for copyrights defined in the Internet Standards process 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

Send Us Your Comments	xiii
Preface.....	xv
1 Introduction	
About Oracle Internet Directory Software Developer's Kit Release 9.2.....	1-2
Components of the Oracle Internet Directory Software Developer's Kit.....	1-2
Other Components of Oracle Internet Directory.....	1-2
Operating Systems Supported	1-3
Part I Standard LDAP APIs	
2 Concepts	
History of LDAP	2-2
Overview of LDAP Models	2-2
LDAP Naming Model.....	2-2
LDAP Information Model	2-4
LDAP Functional Model.....	2-5
LDAP Security Model.....	2-6
About the Oracle Internet Directory API.....	2-11
Initializing an LDAP Session	2-14
Initializing the Session by Using the C API.....	2-14
Initializing the Session by Using DBMS_LDAP.....	2-15
LDAP Session Handle Options in the C API.....	2-16

Enabling Authentication to a Directory Server	2-16
Enabling Authentication to a Directory Server by Using the C API.....	2-16
Enabling Authentication to a Directory Server by Using DBMS_LDAP.....	2-17
Searching by Using DBMS_LDAP	2-18
Flow of Search-Related Operations.....	2-19
Search Scope	2-22
Filters	2-23
Enabling Session Termination by Using DBMS_LDAP	2-24

3 C API for Oracle Internet Directory

About the Oracle Internet Directory C API	3-2
Oracle Internet Directory SDK C API SSL Extensions	3-2
C API Reference	3-4
Summary of LDAP C API.....	3-4
Functions.....	3-8
Initializing an LDAP Session	3-9
LDAP Session Handle Options.....	3-10
Working With Controls.....	3-15
Authenticating to the Directory.....	3-17
Closing the Session	3-20
Performing LDAP Operations	3-21
Abandoning an Operation.....	3-42
Obtaining Results and Peeking Inside LDAP Messages.....	3-43
Handling Errors and Parsing Results	3-46
Stepping Through a List of Results.....	3-49
Parsing Search Results	3-50
Sample C API Usage	3-61
C API Usage with SSL.....	3-61
C API Usage Without SSL	3-62
Building Applications with the C API	3-63
Required Header Files and Libraries	3-63
Building a Sample Search Tool	3-63
Dependencies and Limitations	3-76

4 The DBMS_LDAP PL/SQL Package

About the DBMS_LDAP Package	4-2
Building Applications with DBMS_LDAP	4-2
Dependencies and Limitations	4-2
DBMS_LDAP Sample Programs	4-3
DBMS_LDAP Reference	4-3
Summary of Subprograms	4-3
Exception Summary	4-6
Data-Type Summary	4-9
Subprograms	4-10

Part II Oracle Extensions to LDAP APIs

5 Overview of Oracle Extensions

The LDAP Access Model	5-2
Application Installation Logic	5-3
Application Startup and Bootstrap Logic	5-3
Application Runtime Logic	5-3
Application Shutdown Logic.....	5-4
Application Deinstallation Logic	5-4
Entities Modeled in LDAP	5-4
Users	5-5
Groups.....	5-5
Subscribers.....	5-5
API Enhancements: Overview & Usage Model	5-6
API Enhancements: Assumptions	5-6
System Placement	5-7
API Enhancements Functional Categorization	5-7
API Enhancements Usage Model	5-8
Installation and First Use	5-11

6 Java API for Oracle Internet Directory

Class Descriptions	6-2
User Class	6-2

Subscriber Class	6-3
Group Class	6-4
PropertySetCollection, PropertySet, and Property Classes	6-5
Classes	6-6
oracle.ldap.util.Base64.....	6-7
oracle.ldap.util.Group	6-9
oracle.ldap.util.Guid.....	6-11
oracle.ldap.util.LDIF.....	6-14
oracle.ldap.util.LDIFAttribute	6-16
oracle.ldap.util.LDIFMigration.....	6-26
oracle.ldap.util.LDIFReader	6-30
oracle.ldap.util.LDIFRecord.....	6-33
oracle.ldap.util.LDIFSubstitute.....	6-37
oracle.ldap.util.LDIFWriter	6-38
oracle.ldap.util.Property	6-43
oracle.ldap.util.PropertySet.....	6-44
oracle.ldap.util.PropertySetCollection.....	6-47
oracle.ldap.util.Subscriber	6-49
oracle.ldap.util.User	6-52
oracle.ldap.util.Util.....	6-56
java.util.Hashtable getAllDASUrl(DirContext ctx)	6-59
oracle.ldap.util.jndi.ConnectionUtil.....	6-67
Exceptions	6-69
oracle.ldap.util.AcctIPLockedException	6-70
oracle.ldap.util.AcctTotallyLockedException.....	6-70
oracle.ldap.util.AuthFailureException	6-71
oracle.ldap.util.AuthPasswdChangeWarningException	6-71
oracle.ldap.util.AuthPasswdExpiredException	6-71
oracle.ldap.util.GeneralErrorException.....	6-72
oracle.ldap.util.InvalidLDIFRecordException.....	6-73
oracle.ldap.util.InvalidParameterException	6-74
oracle.ldap.util.InvalidRootOrclctxException	6-75
oracle.ldap.util.InvalidSubscriberOrclctxException	6-76
oracle.ldap.util.LoginPolicyFailureException	6-76
oracle.ldap.util.MigrationException	6-77

oracle.ldap.util.MultipleSubscriberException	6-78
oracle.ldap.util.MultipleUserException	6-79
oracle.ldap.util.NoGroupMembersException	6-79
oracle.ldap.util.NoRootOrclctxException	6-80
oracle.ldap.util.NoSubscriberOrclctxException	6-81
oracle.ldap.util.NoSuchGroupException	6-82
oracle.ldap.util.NoSuchSubscriberException	6-83
oracle.ldap.util.NoSuchUserException	6-84
oracle.ldap.util.ParameterException.....	6-84
oracle.ldap.util.PasswdExpiredException	6-85
oracle.ldap.util.SetPropertiesException	6-86
oracle.ldap.util.SubscriberNotFoundException.....	6-86
oracle.ldap.util.UtilException	6-87

7 The DBMS_LDAP_UTL PL/SQL Package

Introduction	7-2
DBMS_LDAP_UTL Reference	7-2
Summary of Subprograms	7-3
User-Related Subprograms	7-4
Group-Related Subprograms.....	7-22
Subscriber-Related Subprograms.....	7-29
Property-Related Subprograms.....	7-34
Miscellaneous Subprograms	7-35
Function Return Code Summary	7-45
Data-Type Summary	7-47

8 Developing Provisioning-Integrated Applications

Prerequisite Knowledge	8-2
Development Usage Model for Provisioning Integration	8-2
Development Tasks for Provisioning Integration	8-4
Application Installation	8-5
User Creation and Enrollment.....	8-5
User Deletion.....	8-5
Application Deinstallation	8-7
Provisioning Event Interface Description	8-7

LDAP_NOTIFY Function Definitions	8-9
FUNCTION event_ntfy.....	8-11

9 Oracle Internet Directory Server Plug-in Framework

Introduction	9-2
Prerequisite Knowledge	9-2
Concepts	9-2
About Directory Server Plug-ins	9-2
About Server Plug-in Framework	9-3
Operation-Based Plug-ins Supported in Oracle Internet Directory	9-4
Requirements	9-6
Designing Plug-ins	9-6
Creating Plug-ins	9-7
Compiling Plug-ins.....	9-10
Registering Plug-ins	9-10
Managing Plug-ins.....	9-13
Enabling and Disabling Plug-ins.....	9-14
Exception Handling.....	9-14
Plug-in LDAP API	9-16
Plug-in and Replication	9-17
Plug-in and DB Tools	9-17
Security.....	9-17
Plug-in LDAP API Specifications	9-18
Usage Model and Examples	9-18
Example 1: Search Query Logging.....	9-18
Example 2: Synchronizing Two DITs	9-21
Type Definition & Usage Model	9-23
Database Object Type Definitions	9-23
Plug-in Module Interface Specifications.....	9-24
LDAP Server Error Code Reference	9-28

Part III Appendixes

A Command-Line Tools Syntax

LDAP Data Interchange Format (LDIF) Syntax	10-2
Entry-Management Command-Line Tools	10-4
ldapadd Syntax	10-4
ldapaddmt Syntax	10-7
ldapbind Syntax	10-9
ldapdelete Syntax	10-10
ldapmoddn Syntax	10-11
ldapsearch Syntax.....	10-13
Attribute-Management Command-Line Tools	10-18
The Catalog Management Tool	10-18
ldapcompare Syntax.....	10-19
ldapmodify Syntax	10-22
ldapmodifymt Syntax	10-27
Provisioning Subscription Tool	10-29

B Sample Usage

DBMS_LDAP Sample Code	A-2
Using DBMS_LDAP from a Database Trigger	A-2
Using DBMS_LDAP for a Search	A-10
DBMS_LDAP_UTL Sample Code	A-15
Example: User-Related Functions	A-15
Example: Property-Related Subprograms	A-20
Example: Subscriber-Related Functions	A-25
Example: Group-Related Functions.....	A-28
Java Sample Code	A-34
User Class Sample Code	A-34
Subscriber Class Sample Code.....	A-37
Group Class Sample Code.....	A-39
Print Sample Code.....	A-41

Glossary

Index

Send Us Your Comments

Oracle Internet Directory Application Developer's Guide, Release 9.2

Part No. A96577-01

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: infodev_us@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](#)
- [Organization](#)
- [Related Documentation](#)
- [Conventions](#)
- [Documentation Accessibility](#)

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

Chapter 1, "Introduction"

Briefly describes the intended audience and components of Oracle Internet Directory Software Developer's Kit Release 9.2. It also lists the other components of Oracle Internet Directory and the platforms it supports.

Chapter 2, "Concepts"

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, "C API for Oracle Internet Directory"

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

Chapter 4, "The DBMS_LDAP PL/SQL Package"

Introduces the PL/SQL API, which is contained in a PL/SQL package called DBMS_LDAP. It also contains examples of how to use it.

Chapter 5, "Overview of Oracle Extensions"

This chapter explains how to directory-enable your applications.

Chapter 6, "Java API for Oracle Internet Directory"

This chapter contains reference material for the Java API for Oracle Internet Directory.

Chapter 7, "The DBMS_LDAP_UTL PL/SQL Package"

This chapter introduces the DBMS_LDAP_UTL Package, which contains Oracle Extension utility functions.

Chapter 8, "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 Platform. These applications can be either legacy or third-party applications that are based on the Oracle platform.

Chapter 9, "Oracle Internet Directory Server Plug-in Framework"

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

Appendix A, "Command-Line Tools Syntax"

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.

Glossary

Related Documentation

For more information, see these Oracle resources:

- Oracle9i documentation set, especially
 - *Oracle Internet Directory Administrator's Guide*.
 - *PL/SQL User's Guide and Reference*

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, especially:
 - <http://www.ietf.org> for the IETF home page
 - <http://www.ietf.org/html.charters/ldapext-charter.html> for the ldapext charter and LDAP drafts)
 - <http://www.ietf.org/html.charters/ldup-charter.html> for the LDUP charter and drafts
 - <http://www.ietf.org/rfc/rfc2254.txt>, "The String Representation of LDAP Search Filters"
 - <http://www.ietf.org/rfc/rfc1823.txt>, "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 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 .
<i>Italics</i>	Italic typeface indicates book titles or emphasis.	<i>Oracle9i Database Concepts</i> Ensure that the recovery catalog and target database do <i>not</i> 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.

Convention	Meaning	Example
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 <code>sqlplus</code> to open SQL*Plus. The password is specified in the <code>orapwd</code> file. Back up the datafiles and control files in the <code>/disk1/oracle/dbs</code> directory. The <code>department_id</code> , <code>department_name</code> , and <code>location_id</code> columns are in the <code>hr.departments</code> table. Set the <code>QUERY_REWRITE_ENABLED</code> initialization parameter to <code>true</code> . Connect as <code>oe</code> user. The <code>JRepUtil</code> class implements these methods.
<i>lowercase monospace (fixed-width font) italic</i>	Lowercase monospace italic font represents placeholders or variables.	You can specify the <i>parallel_clause</i> . Run <code>Uold_release.SQL</code> where <i>old_release</i> 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.	<code>DECIMAL (digits [, precision])</code>
{ }	Braces enclose two or more items, one of which is required. Do not enter the braces.	<code>{ENABLE DISABLE}</code>
	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.	<code>{ENABLE DISABLE}</code> <code>[COMPRESS NOCOMPRESS]</code>

Convention	Meaning	Example
...	Horizontal ellipsis points indicate either: <ul style="list-style-type: none"> 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 	<pre>CREATE TABLE ... AS subquery; SELECT col1, col2, ... , coln FROM employees;</pre>
.	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	
Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	<pre>acctbal NUMBER(11,2); acct CONSTANT NUMBER(4) := 3;</pre>
<i>Italics</i>	Italicized text indicates placeholders or variables for which you must supply particular values.	<pre>CONNECT SYSTEM/system_password DB_NAME = database_name</pre>
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.	<pre>SELECT last_name, employee_id FROM employees; SELECT * FROM USER_TABLES; DROP TABLE hr.employees;</pre>
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.	<pre>SELECT last_name, employee_id FROM employees; sqlplus hr/hr CREATE USER mjones IDENTIFIED BY ty3MU9;</pre>

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.

Introduction

This chapter briefly describes the intended audience and components of Oracle Internet Directory Software Developer's Kit Release 9.2. 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 Release 9.2](#)
- [Components of the Oracle Internet Directory Software Developer's Kit](#)
- [Other Components of Oracle Internet Directory](#)
- [Operating Systems Supported](#)

About Oracle Internet Directory Software Developer's Kit Release 9.2

Oracle Internet Directory SDK Release 9.2 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 Release 9.2 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

Other Components of Oracle Internet Directory

The following components of Oracle Internet Directory Release 9.2, 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, both servers and clients, support these operating systems:

- Sun Solaris
- Microsoft Windows
 - Windows NT 4.0
 - Windows 98
 - Windows 2000
- HPUX
- AIX
- Compaq TRU64
- Intel Solaris
- SGI
- DGUX
- UNIXWARE

Part I

Standard LDAP APIs

Part I explains the core LDAP APIs, and includes some of the basic LDAP programming concepts, IETF C API information, and PL/SQL API information. It contains these chapters:

- [Chapter 2, "Concepts"](#)
- [Chapter 3, "C API for Oracle Internet Directory"](#)
- [Chapter 4, "The DBMS_LDAP PL/SQL Package"](#)

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. The concepts acquired in this section make it easier to understand the API details.

This chapter contains these topics:

- [History of LDAP](#)
- [Overview of LDAP Models](#)
- [About the Oracle Internet Directory API](#)
- [Initializing an LDAP Session](#)
- [LDAP Session Handle Options in the C API](#)
- [Enabling Authentication to a Directory Server](#)
- [Searching by Using DBMS_LDAP](#)
- [Enabling Session Termination by Using DBMS_LDAP](#)

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:

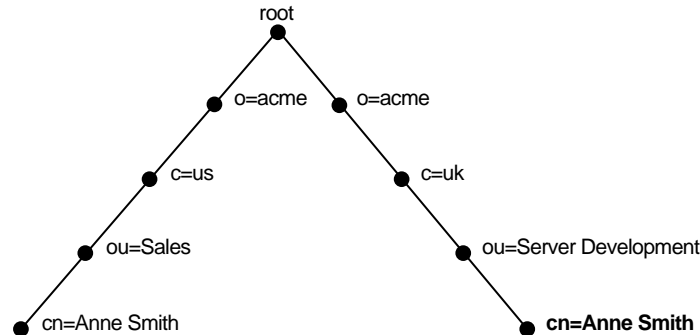
- [LDAP Naming Model](#)
- [LDAP Information Model](#)
- [LDAP Functional Model](#)
- [LDAP Security Model](#)

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

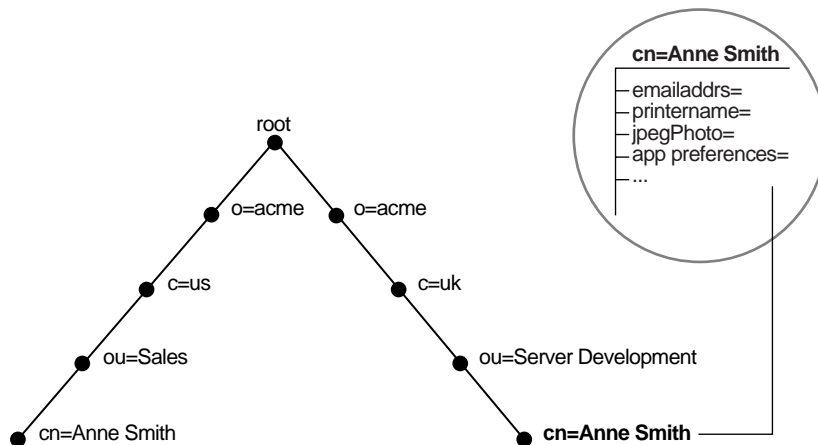
The LDAP information model determines the form and character of information in the directory. It is centered around entries, which are composed of attributes. In a directory, each collection of information about an object is called an **entry**. For example, a typical telephone directory includes entries 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 `emailaddr`, `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:

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:

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.

Access control information 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

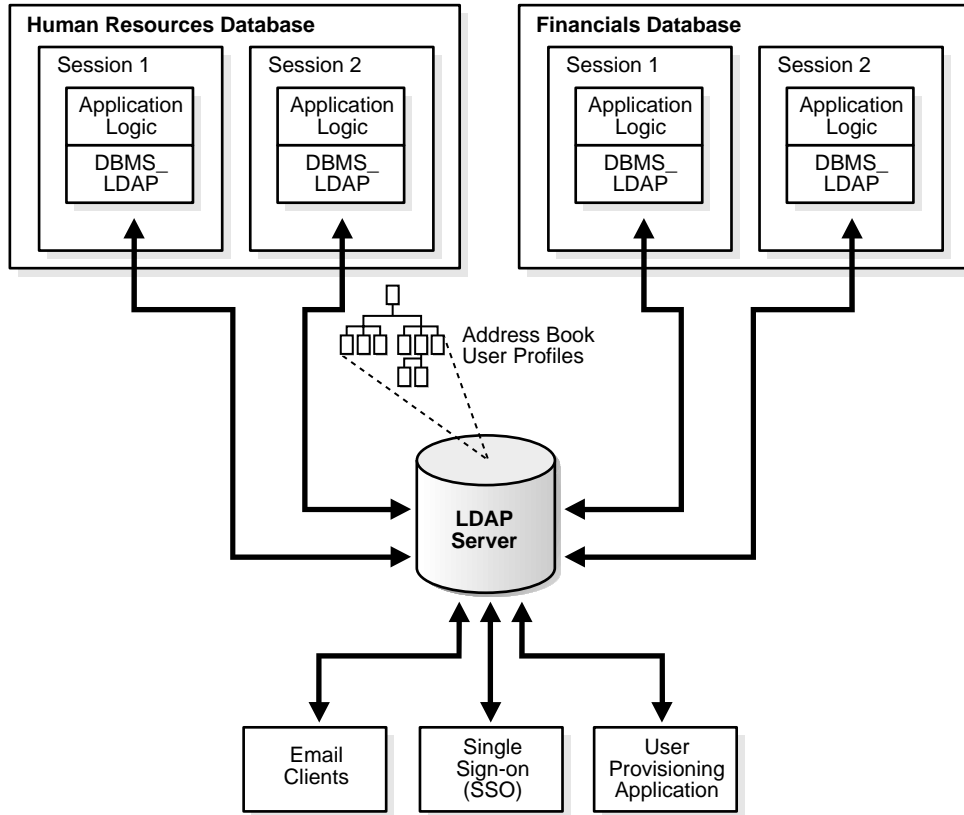
About the Oracle Internet Directory API

The Oracle Internet Directory API is available as a C API and as a PL/SQL API.

The PL/SQL API is contained in a PL/SQL package called `DBMS_LDAP`. This 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.

Figure 2-3 illustrates the overall placement of the DBMS_LDAP API in the runtime environment of a client.

Figure 2-3 Applications Sharing LDAP Server Data



As Figure 2-3 shows, the API allows multiple different applications—in this example, Human Resources and Financials—to share employee address book information and user profiles by using an LDAP server.

Storing such information in an LDAP server enables other non-database applications that are LDAP-enabled to retrieve the same information. In Figure 2-3, the Email Clients application uses the same employee address book data to find the employee for a given email address. Because LDAP offers a centralized repository of user information, the same information can be used for Single Sign-On applications and other enterprise-wide user provisioning applications.

In summary, the Oracle Internet Directory API enables Oracle database applications to:

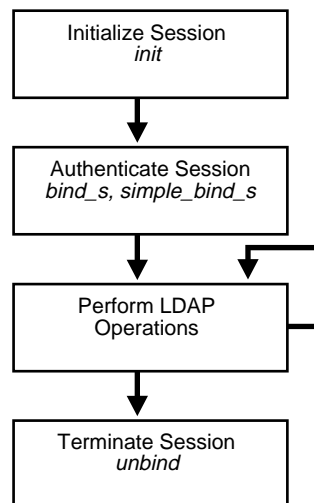
- Read from the LDAP server information that is published by other programs in the enterprise
- Publish in the LDAP server new information that can be used later by the same application or other applications
- Modify or update existing information in the LDAP server based on certain pre-defined conditions

Typically, an application or trigger uses the functions in the API in four simple 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 2-4 illustrates these steps.

Figure 2-4 Steps in Typical DBMS_LDAP Usage



The following sections explain the important features of the API with respect to each of these steps.

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.

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 2–1 Parameters for `ldap_init()`

Parameter	Description
hostname	<p>Contains a space-separated list of hostnames or dotted strings representing the IP address of hosts running an LDAP server to connect to. Each hostname 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.</p> <p>Note: A suitable representation for including a literal IPv6[10] address in the hostname parameter is desired, but has not yet been determined or implemented in practice.</p>
portno	<p>Contains the TCP port number to connect to. The default LDAP port of 389 can be obtained by supplying the constant <code>LDAP_PORT</code>. If a host includes a port number then this parameter is ignored.</p>

`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.

Note that if you connect to an LDAPv2 server, one of the LDAP bind calls described below **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.

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 hostname 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. 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.

LDAP Session Handle Options in the C API

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 below.

`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.

Enabling Authentication to a Directory Server

Before initiating any of the LDAP operations, an individual or application seeking to perform operations against an LDAP server must be authenticated.

Enabling Authentication to a Directory Server by Using the C API

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 OID 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.

Enabling Authentication to a Directory Server by Using DBMS_LDAP

The functions `simple_bind_s` and `bind_s` enable 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 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. 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. The function that initiates SSL communications is called `open_ssl` and its syntax is:

```
FUNCTION open_ssl(ld IN SESSION, sslwrl IN VARCHAR2,
    sslwalletpasswd IN VARCHAR2, sslauth IN PLS_INTEGER)
    RETURN PLS_INTEGER;
```

The `open_ssl` function should be called immediately after the call to `init` to secure the LDAP TCP/IP connection from eavesdroppers. Authentication is done implicitly by using the credentials in the certificate stored in the wallet.

See Also: *Oracle Advanced Security Administrator's Guide* for instructions on using the Oracle Wallet Manager

The following PL/SQL code snippet shows a typical usage of the initialization, authentication, and cleanup functions that were 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);
    --Bind 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 by Using DBMS_LDAP

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 release of 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.

There are two functions available for initiating searches in the DBMS_LDAP API:

- `DBMS_LDAP.search_s()`
- `DBMS_LDAP.search_st()`

The only difference between the two is that `search_st()` uses a client side timeout to stop the search if it exceeds a certain elapsed time limit. 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 these arguments:

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` and `search_st`, several support functions in the API help in retrieving search results. These are highlighted in the following section.

Flow of Search-Related Operations

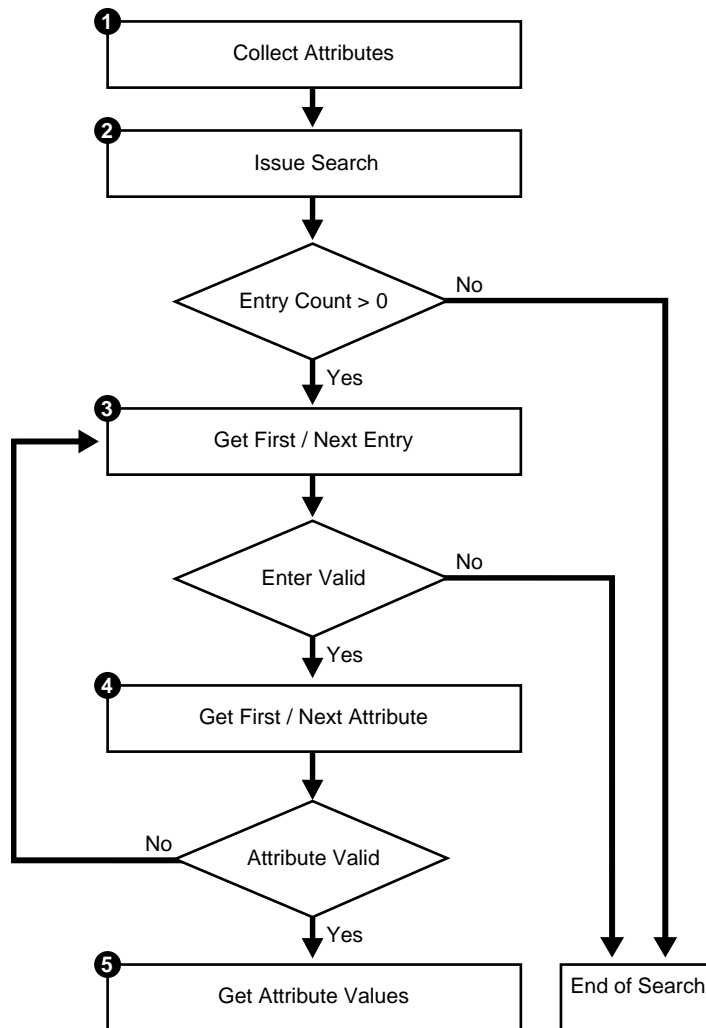
The programming work required to initiate a typical search operation and retrieve results can be broken down into the following steps:

1. Decide the attributes that need to be returned, and compose them into 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`).
4. For the entry obtained in Step 3, get an attribute (using `DBMS_LDAP.first_attribute` or `DBMS_LDAP.next_attribute`).
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`)
6. Repeat Step 4 until all attributes of the entry are examined
7. Repeat Step 3 until there are no more entries

Figure 2-5 illustrates the above steps in more detail.

Figure 2-5 Flow of Search-Related Operations



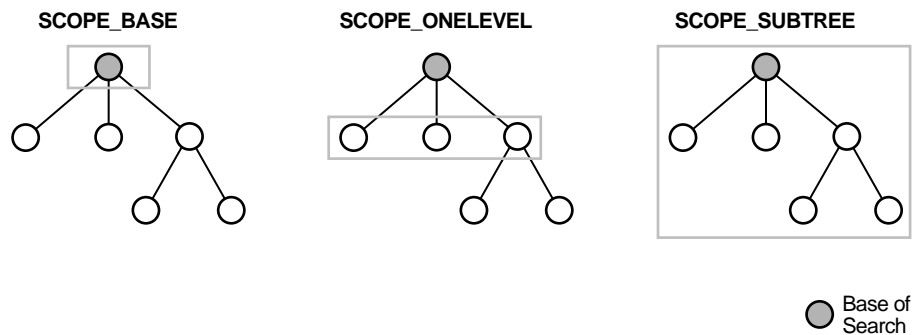
Search Scope

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:

- **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 2-6 illustrates the difference between the three scope options.

Figure 2-6 The Three Scope Options



In Figure 2-6, 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 **Internet Engineering Task Force (IETF) RFC 1960**. 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 2–2 Search Filters

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

The basic filters in [Table 2–2](#) 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 2–3](#) summarizes the fundamental Boolean operations:

Table 2–3 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.

Enabling Session Termination 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`.

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](#)
- [C API Reference](#)
- [Sample C API Usage](#)
- [Building Applications with the C API](#)
- [Dependencies and Limitations](#)

About the Oracle Internet Directory C API

The Oracle Internet Directory SDK C API is based on:

- LDAP Version 3 C API
- Oracle extensions to support SSL

You can use the Oracle Internet Directory API Release 9.2 in the following modes:

- SSL—All communication secured using SSL
- Non-SSL—Client/server communication not secure

The API uses TCP/IP to connect to an LDAP 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 3-61 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(Socketbuf *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, all subsequent communication happens over a secure connection.

Argument	Description
<code>sb</code>	Socket buffer handle returned by the <code>ldap_open</code> call as part of LDAP handle.
<code>sslwallet</code>	Location of the user wallet.
<code>sslwalletpasswd</code>	Password required to use the wallet.
<code>sslauthmode</code>	<p>SSL authentication mode user wants to use. Possible values are:</p> <ul style="list-style-type: none"> ■ <code>GSLC_SSL_NO_AUTH</code>—No authentication required ■ <code>GSLC_SSL_ONEWAY_AUTH</code>—Only server authentication required. ■ <code>GSLC_SSL_TWOWAY_AUTH</code>—Both server and client authentication required. <p>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 <code>ldap_err2string</code>.</p>

See Also: See "[Sample C API Usage](#)" on page 3-61

Wallet Support

To use the SSL feature, both the server and the client may require wallets, depending on which authentication mode is being used. Release 9.2 of the API supports only Oracle Wallet. You can create wallets using Oracle Wallet Manager.

C API Reference

This section contains these topics:

- [Summary of LDAP C API](#)
- [Functions](#)
- [Initializing an LDAP Session](#)
- [LDAP Session Handle Options](#)
- [Working With Controls](#)
- [Authenticating to the Directory](#)
- [Closing the Session](#)
- [Performing LDAP Operations](#)
- [Abandoning an Operation](#)
- [Obtaining Results and Peeking Inside LDAP Messages](#)
- [Handling Errors and Parsing Results](#)
- [Stepping Through a List of Results](#)
- [Parsing Search Results](#)
- [C API Usage with SSL](#)
- [C API Usage Without SSL](#)

Summary of LDAP C API

Table 3–1 DBMS_LDAP API Subprograms

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

Table 3–1 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
ldap_delete_ext	Delete an entry from the directory
ldap_delete_ext_s	
ldap_delete	
ldap_delete_s	
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
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

Table 3–1 DBMS_LDAP API Subprograms

Function or Procedure	Description
ldap_get_values	Get the string values of an attribute
ldap_get_values_len	Get the binary values of an attribute
ldap_init	Open a connection to an LDAP server
ldap_open	
ldap_memfree()	Free memory allocated by an LDAP API function call
ldap_modify_ext	Modify an entry in the directory
ldap_modify_ext_s	
ldap_modify	
ldap_modify_s	
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	Prints the message supplied in message.
DEPRECATED	
ldap_rename	Modify the RDN of an entry in the directory
ldap_rename_s	
ldap_result2error	Returns the error code from result message.
DEPRECATED	
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	

Table 3–1 DBMS_LDAP API Subprograms

Function or Procedure	Description
<code>ldap_search_ext</code>	Search the directory
<code>ldap_search_ext_s</code>	
<code>ldap_search</code>	
<code>ldap_search_s</code>	
<code>ldap_search_st</code>	Search the directory with a timeout value
<code>ldap_set_option</code>	Set the value of these parameters
<code>ldap_simple_bind</code>	Simple authentication to an LDAP server
<code>ldap_simple_bind_s</code>	
<code>ldap_unbind_ext</code>	End an LDAP session
<code>ldap_unbind</code>	
<code>ldap_unbind_s</code>	
<code>ldap_value_free</code>	Free the memory allocated for the string values of an attribute
<code>ldap_value_free_len</code>	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:
<http://www.ietf.org/rfc/rfc1823.txt> for a more detailed explanation of these calls

Functions

This section contains these topics:

- [Initializing an LDAP Session](#)
- [LDAP Session Handle Options](#)
- [Authenticating to the Directory](#)
- [Working With Controls](#)
- [Closing the Session](#)
- [Performing LDAP Operations](#)
- [Abandoning an Operation](#)
- [Obtaining Results and Peeking Inside LDAP Messages](#)
- [Handling Errors and Parsing Results](#)
- [Stepping Through a List of Results](#)
- [Parsing Search Results](#)

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 3–2 Parameters for Initializing an LDAP Session

Parameter	Description
hostname	<p>Contains a space-separated list of hostnames or dotted strings representing the IP address of hosts running an LDAP server to connect to. Each hostname 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.</p> <p>Note: A suitable representation for including a literal IPv6[10] address in the hostname parameter is desired, but has not yet been determined or implemented in practice.</p>
portno	<p>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.</p>

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 3–3 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 <code>ldap_init()</code> or <code>ldap_open()</code> 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 3–4 . 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 <code>char **</code> and <code>LDAPControl **</code> , a copy of the data that is associated with the LDAP session ld is returned; callers should dispose of the memory by calling <code>ldap_memfree()</code> or <code>ldap_controls_free()</code> , depending on the type of data returned.

Table 3–3 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.

Table 3–4 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.
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.

Table 3–4 Constants

Constant	Type for invalue parameter	Type for outvalue parameter	Description
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.
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).

Table 3–4 Constants

Constant	Type for invalue parameter	Type for outvalue parameter	Description
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 3-15
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 3-15
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 hostname parameter to ldap_init() for the allowed syntax.
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 have the following meanings:

Table 3–5 *Fields in `ldapcontrol` Structure*

Field	Description
<code>ldctl_oid</code>	The control type, represented as a string.

Table 3–5 Fields in Ldapcontrol Structure

Field	Description
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 or 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 3-10. 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.

A Client Control That Governs Referral Processing As described previously in "[LDAP Session Handle Options](#)" on page 3-10, applications can enable and disable automatic chasing of referrals on a session-wide basis 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 3–6 Parameters for Authenticating to the Directory

Parameter	Description
ld	The session handle
dn	The name of the entry to bind as
mechanism	Either LDAP_SASL_SIMPLE (NULL) to get simple authentication, or a text string identifying the SASL method
cred	The credentials with which to authenticate. Arbitrary credentials can be passed using this parameter. The format and content of the credentials depends on the setting of the mechanism parameter.
passwd	For ldap_simple_bind(), the password to compare to the entry's userPassword attribute
serverctrls	List of LDAP server controls
clientctrls	List of client controls
msgidp	This result parameter will be set to the message id of the request if the ldap_sasl_bind() call succeeds
servercredp	This result parameter will be filled in with the credentials passed back by the server for mutual authentication, if given. An allocated berval structure is returned that SHOULD be disposed of by calling ber_bvfree(). NULL SHOULD be passed to ignore this field.

Usage Notes

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

The ldap_sasl_bind() function initiates an asynchronous bind operation and returns the constant LDAP_SUCCESS if the request was successfully sent, or another LDAP error code if not. If successful, ldap_sasl_bind() places the message id of the request

in `*msgidp`. A subsequent call to `ldap_result()`, described in "[ldap_result](#)" on page 3-43, 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 3-46 for more information about possible errors and how to interpret them.

Closing the Session

`ldap_unbind_ext`

`ldap_unbind`

`ldap_unbind_s`

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

Syntax

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


Parameters

Table 3–7 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

ldap_search_ext

ldap_search_ext_s

ldap_search

ldap_search_s

ldap_search_st

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

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 3–8 Parameters for Search Operations

Parameter	Description
ld	The session handle.
base	The dn of the entry at which to start the search.
scope	One of LDAP_SCOPE_BASE (0x00), LDAP_SCOPE_ONELEVEL (0x01), or LDAP_SCOPE_SUBTREE (0x02), indicating the scope of the search.
filter	A character string representing the search filter. The value NULL can be passed to indicate that the filter "(objectclass=*)" which matches all entries is to be used. Note that if the caller of the API is using LDAPv2, only a subset of the filter functionality can be successfully used.

Table 3–8 Parameters for Search Operations

Parameter	Description
attrs	A NULL-terminated array of strings indicating which attributes to return for each matching entry. Passing NULL for this parameter causes all available user attributes to be retrieved. The special constant string LDAP_NO_ATTRS ("1.1") MAY be used as the only string in the array to indicate that no attribute types are to be returned by the server. The special constant string LDAP_ALL_USER_ATTRS ("*") can be used in the attrs array along with the names of some operational attributes to indicate that all user attributes plus the listed operational attributes are to be returned.
attrsonly	A boolean value that MUST be zero if both attribute types and values are to be returned, and nonzero if only types are wanted.
timeout	For the ldap_search_st() function, this specifies the local search timeout value (if it is NULL, the timeout is infinite). If a zero timeout (where tv_sec and tv_usec are both zero) is passed, API implementations SHOULD return LDAP_PARAM_ERROR. For the ldap_search_ext() and ldap_search_ext_s() functions, the timeout parameter specifies both the local search timeout value and the operation time limit that is sent to the server within the search request. Passing a NULL value for timeout causes the global default timeout stored in the LDAP session handle (set by using ldap_set_option() with the LDAP_OPT_TIMELIMIT parameter) to be sent to the server with the request but an infinite local search timeout to be used. If a zero timeout (where tv_sec and tv_usec are both zero) is passed in, API implementations SHOULD return LDAP_PARAM_ERROR. If a zero value for tv_sec is used but tv_usec is nonzero, an operation time limit of 1 SHOULD be passed to the LDAP server as the operation time limit. For other values of tv_sec, the tv_sec value itself SHOULD be passed to the LDAP server.
sizelimit	For the ldap_search_ext() and ldap_search_ext_s() calls, this is a limit on the number of entries to return from the search. A value of LDAP_NO_LIMIT (0) means no limit.
res	For the synchronous calls, this is a result parameter which will contain the results of the search upon completion of the call. If no results are returned, *res is set to NULL.
serverctrls	List of LDAP server controls.
clientctrls	List of client controls.

Table 3–8 Parameters for Search Operations

Parameter	Description
msgidp	<p>This result parameter will be set to the message id of the request if the <code>ldap_search_ext()</code> call succeeds. There are three options in the session handle <code>ld</code> which potentially affect how the search is performed. They are:</p> <ul style="list-style-type: none"> ▪ <code>LDAP_OPT_SIZELIMIT</code>—A limit on the number of entries to return from the search. A value of <code>LDAP_NO_LIMIT (0)</code> means no limit. Note that the value from the session handle is ignored when using the <code>ldap_search_ext()</code> or <code>ldap_search_ext_s()</code> functions. ▪ <code>LDAP_OPT_TIMELIMIT</code>—A limit on the number of seconds to spend on the search. A value of <code>LDAP_NO_LIMIT (0)</code> means no limit. Note that the value from the session handle is ignored when using the <code>ldap_search_ext()</code> or <code>ldap_search_ext_s()</code> functions. ▪ <code>LDAP_OPT_DEREF</code>—One of <code>LDAP_DEREF_NEVER (0x00)</code>, <code>LDAP_DEREF_SEARCHING (0x01)</code>, <code>LDAP_DEREF_FINDING (0x02)</code>, or <code>LDAP_DEREF_ALWAYS (0x03)</code>, specifying how aliases are handled during the search. The <code>LDAP_DEREF_SEARCHING</code> value means aliases are dereferenced during the search but not when locating the base object of the search. The <code>LDAP_DEREF_FINDING</code> 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 3-43, 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 3-43, 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 3-46 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 3–9 Parameters for Compare Operations

Parameter	Description
ld	The session handle.
dn	The name of the entry to compare against.
attr	The attribute to compare against.
bvalue	The attribute value to compare against those found in the given entry. This parameter is used in the extended routines and is a pointer to a struct berval so it is possible to compare binary values.
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 3-43, 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 3-43, 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 3-46 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 3–10 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

The fields in the LDAPMod structure have the following meanings:

Table 3–11 Fields in LDAPMod Structure

Field	Description
mod_op	The modification operation to perform. It MUST be one of LDAP_MOD_ADD (0x00), LDAP_MOD_DELETE (0x01), or LDAP_MOD_REPLACE (0x02). This field also indicates the type of values included in the mod_vals union. It is logically ORed with LDAP_MOD_BVALUES (0x80) to select the mod_bvalues form. Otherwise, the mod_values form is used.
mod_type	The type of the attribute to modify.

Table 3–11 Fields in LDAPMod Structure

Field	Description
mod_vals	The values (if any) to add, delete, or replace. Only one of the mod_values or mod_bvalues variants can be used, selected by ORing the mod_op field with the constant LDAP_MOD_BVALUES. mod_values is a NULL-terminated array of zero-terminated strings and mod_bvalues is a NULL-terminated array of berval structures that can be used to pass binary values such as images.

Usage Notes

For LDAP_MOD_ADD modifications, the given values are added to the entry, creating the attribute if necessary.

For LDAP_MOD_DELETE modifications, the given values are deleted from the entry, removing the attribute if no values remain. If the entire attribute is to be deleted, the mod_vals field can be set to NULL.

For LDAP_MOD_REPLACE modifications, the attribute will have the listed values after the modification, having been created if necessary, or removed if the mod_vals field is NULL. All modifications are performed in the order in which they are listed.

The ldap_modify_ext() function initiates an asynchronous modify operation and returns the constant LDAP_SUCCESS if the request was successfully sent, or another LDAP error code if not. If successful, ldap_modify_ext() places the message id of the request in *msgidp. A subsequent call to ldap_result(), described in "[ldap_result](#)" on page 3-43, 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 3-43, 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 3-46 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 3–12 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.

Table 3–12 Parameters for Rename Operations

Parameter	Description
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 3-43, 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 3-46 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 3–13 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 <code>ldap_modify()</code> . The <code>mod_type</code> and <code>mod_vals</code> fields MUST be filled in. The <code>mod_op</code> field is ignored unless ORed with the constant <code>LDAP_MOD_BVALUES</code> , used to select the <code>mod_bvalues</code> case of the <code>mod_vals</code> 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 <code>ldap_add_ext()</code> 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 3-43, 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 3-43, 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 3-46 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 3–14 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 <code>ldap_delete_ext()</code> 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 3-43, 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 3-43, 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 3-46 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 3–15 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 3–15 Parameters for Extended Operations

Parameter	Description
msgidp	This result parameter will be set to the message id of the request if the <code>ldap_extended_operation()</code> 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 <code>ldap_memfree()</code> function. If no OID was returned, <code>*retoidp</code> 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 <code>ber_bvfree()</code> . If no data is returned, <code>*retdatap</code> 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 3-43, 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 3-46 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 3–16 *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 3-46 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 3–17 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_MOVDN (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, 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          **matcheddn,
    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 3–18 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.
matcheddn	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.

Table 3–18 Parameters for Handling Errors and Parsing Results

Parameter	Description
errmsgp	This result parameter will be filled in with the contents of the error message field from the LDAPMessage message. The error message string SHOULD be freed by calling ldap_memfree() which is described later in this document. NULL SHOULD be passed to ignore this field.
referralsp	This result parameter will be filled in with the contents of the referrals field from the LDAPMessage message, indicating zero or more alternate LDAP servers where the request is to be retried. The referrals array SHOULD be freed by calling ldap_value_free() which is described later in this document. NULL SHOULD be passed to ignore this field.
serverctrlsp	This result parameter will be filled in with an allocated array of controls copied out of the LDAPMessage message. The control array SHOULD be freed by calling ldap_controls_free() which was described earlier.
freeit	A boolean that determines whether the res parameter is disposed of or not. Pass any nonzero value to have these routines free res after extracting the requested information. This is provided as a convenience; you can also use ldap_msgfree() to free the result later. If freeit is nonzero, the entire chain of messages represented by res is disposed of.
servercredp	For SASL bind results, this result parameter will be filled in with the credentials passed back by the server for mutual authentication, if given. An allocated berval structure is returned that SHOULD be disposed of by calling ber_bvfree(). NULL SHOULD be passed to ignore this field.
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 `errcode` `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

`ldap_first_message`

`ldap_next_message`

These routines are used to step through the list of messages in a result chain returned by `ldap_result()`. 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 3–19 Parameters for Stepping Through a List of Results

Parameter	Description
ld	The session handle.
res	The result chain, as obtained by a call to one of the synchronous search routines or <code>ldap_result()</code> .
msg	The message returned by a previous call to <code>ldap_first_message()</code> or <code>ldap_next_message()</code> .

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 3–20 Parameters for Retrieving Entries and Continuation References from a Search Result Chain, and for Counting Entries Returned

Parameter	Description
<code>ld</code>	The session handle.
<code>res</code>	The search result, as obtained by a call to one of the synchronous search routines or <code>ldap_result()</code> .
<code>entry</code>	The entry returned by a previous call to <code>ldap_first_entry()</code> or <code>ldap_next_entry()</code> .
<code>ref</code>	The reference returned by a previous call to <code>ldap_first_reference()</code> or <code>ldap_next_reference()</code> .

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 3–21 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 that is described in more detail later in this document in the section "Encoded ASN.1 Value Manipulation".
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 3–22 Parameters for Retrieving and Counting Attribute Values

Parameter	Description
ld	The session handle.
entry	The entry from which to retrieve values, as returned by <code>ldap_first_entry()</code> or <code>ldap_next_entry()</code> .
attr	The attribute whose values are to be retrieved, as returned by <code>ldap_first_attribute()</code> or <code>ldap_next_attribute()</code> , or a caller-supplied string (for example, "mail").
vals	The values returned by a previous call to <code>ldap_get_values()</code> or <code>ldap_get_values_len()</code> .

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 3–23 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 3–24 Parameters for Extracting LDAP Controls from an Entry

Parameters	Description
ld	The session handle.
entry	The entry to extract controls from, as returned by ldap_first_entry() or ldap_next_entry().
serverctrlsp	This result parameter will be filled in with an allocated array of controls copied out of entry. The control array SHOULD be freed by calling ldap_controls_free(). If serverctrlsp is NULL, no controls are returned.

Usage Notes

ldap_get_entry_controls() returns an LDAP error code that indicates whether the reference could be successfully parsed (LDAP_SUCCESS if all goes well).

ldap_parse_reference

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 3–25 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 API both with and without SSL. More complete examples are given in RFC 1823. The sample code for the command line tool to perform LDAP search also demonstrates use of the API in two modes.

This section contains these topics:

- [C API Usage with SSL](#)
- [C API Usage Without SSL](#)

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.

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:

- The header file is located at `$ORACLE_HOME/ldap/public/ldap.h`.
- The library is located at `$ORACLE_HOME/lib/libldapclnt8.a`

Building a Sample Search Tool

The Oracle Internet Directory SDK Release 9.2 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\tturn 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, includeufln, 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
        "KknvtrABLD:s:f:h:b:d:p:F:a:w:l:z:S:"
#else
        "nvtrABLD: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 ( strcasecmp( optarg, "base", 4 ) == 0 ) {
            scope = LDAP_SCOPE_BASE;
        } else if ( strcasecmp( 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 Authentication 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_KREVB41;
    } else {
        authmethod = LDAP_AUTH_KREVB4;
    }
    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, filt patt, value )
LDAP*ld;
char*base;
intscope;
char**attrs;
intattrsonly;
char*filt patt;
char*value;
{
charfilter[ BUFSIZ ], **val;
intrc, first, matches;
LDAPMessage*res, *e;

sprintf( filter, filt patt, 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\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\n", a, sep,

```

```
notascii ? "NOT ASCII" : (char *)bvals[ i ]->bv_val );
    }
}
    }
    gsldedPBerBvecfree( bvals );
}
    }
}

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

    if (( ldif = gslddlDLdifTypeAndValue( type, value, (int)vallen )) == NULL )
    {
        return( -1 );
    }

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

    return( 0 );
}
```

Dependencies and Limitations

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 on 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).

The DBMS_LDAP PL/SQL Package

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. This chapter contains these topics:

- [About the DBMS_LDAP Package](#)
- [Building Applications with DBMS_LDAP](#)
- [Dependencies and Limitations](#)
- [DBMS_LDAP Sample Programs](#)
- [DBMS_LDAP Reference](#)

About the DBMS_LDAP Package

The PL/SQL API in the DBMS_LDAP package is based on the C API described in [Chapter 3, "C API for Oracle Internet Directory"](#).

You can use the Oracle Internet Directory API Release 9.2 in the following modes:

- SSL—All communication secured using SSL
- Non-SSL—Client-to-server communication not secure

The API uses TCP/IP to connect to an LDAP 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.

Building Applications with DBMS_LDAP

To use the PL/SQL LDAP API, you must first 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 SYSDBA using the SQL*Plus command line tool.

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
```

Dependencies and Limitations

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 re-used 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.

DBMS_LDAP Sample Programs

This distribution of Oracle Internet Directory ships with sample programs that illustrate the use of DBMS_LDAP in a relational environment. The samples illustrate the use of the DBMS_LDAP API for the following:

- Synchronizing changes in relational tables to LDAP using database triggers
- Retrieving LDAP entries that match a certain search criteria

The samples are located in the directory `$ORACLE_HOME/ldap/demo/plsql`.

See Also: [Appendix B, "Sample Usage"](#) for a detailed description of the samples

DBMS_LDAP 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 sections before using this section.

This section contains these topics:

- [Summary of Subprograms](#)
- [Exception Summary](#)
- [Data-Type Summary](#)
- [Subprograms](#)

Summary of Subprograms

Table 4-1 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.

Table 4–1 (Cont.) DBMS_LDAP API Subprograms

Function or Procedure	Description
FUNCTION <code>bind_s</code>	The function <code>bind_s</code> can be used to perform complex authentication to the directory server.
FUNCTION <code>unbind_s</code>	The function <code>unbind_s</code> is used for closing an active LDAP session.
FUNCTION <code>compare_s</code>	The function <code>compare_s</code> can be used to test if a particular attribute in a particular entry has a particular value.
FUNCTION <code>search_s</code>	The function <code>search_s</code> 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 <code>search_st</code>	The function <code>search_st</code> 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 <code>first_entry</code>	The function <code>first_entry</code> is used to retrieve the first entry in the result set returned by either <code>search_s</code> or <code>search_st</code> .
FUNCTION <code>next_entry</code>	The function <code>next_entry()</code> is used to iterate to the next entry in the result set of a search operation.
FUNCTION <code>count_entries</code>	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 <code>first_entry()</code> and <code>next_entry()</code> .
FUNCTION <code>first_attribute</code>	The function <code>first_attribute()</code> fetches the first attribute of a given entry in the result set.
FUNCTION <code>next_attribute</code>	The function <code>next_attribute()</code> fetches the next attribute of a given entry in the result set.
FUNCTION <code>get_dn</code>	The function <code>get_dn()</code> retrieves the X.500 distinguished name of given entry in the result set.
FUNCTION <code>get_values</code>	The function <code>get_values()</code> can be used to retrieve all of the values associated for a given attribute in a given entry.

Table 4–1 (Cont.) DBMS_LDAP API Subprograms

Function or Procedure	Description
FUNCTION <code>get_values_len</code>	The function <code>get_values_len()</code> can be used to retrieve values of attributes that have a 'Binary' syntax.
FUNCTION <code>delete_s</code>	This function can be used to remove a leaf entry in the LDAP Directory Information Tree.
FUNCTION <code>modrdn2_s</code>	The function <code>modrdn2_s()</code> can be used to rename the relative distinguished name of an entry.
FUNCTION <code>err2string</code>	The function <code>err2string()</code> can be used to convert an LDAP error code to string in the local language in which the API is operating.
FUNCTION <code>create_mod_array</code>	The function <code>create_mod_array()</code> allocates memory for array modification entries that will be applied to an entry using the <code>modify_s()</code> functions.
PROCEDURE <code>populate_mod_array (String Version)</code>	Populates one set of attribute information for add or modify operations. This procedure call has to happen after <code>DBMS_LDAP.create_mod_array()</code> is called.
PROCEDURE <code>populate_mod_array (Binary Version)</code>	Populates one set of attribute information for add or modify operations. This procedure call has to happen after <code>DBMS_LDAP.create_mod_array()</code> is called.
FUNCTION <code>modify_s</code>	Performs a synchronous modification of an existing LDAP directory entry. Before calling <code>add_s</code> , we have to call <code>DBMS_LDAP.creat_mod_array ()</code> and <code>DBMS_LDAP.populate_mod_array()</code> first.
FUNCTION <code>add_s</code>	Adds a new entry to the LDAP directory synchronously. Before calling <code>add_s</code> , we have to call <code>DBMS_LDAP.creat_mod_array ()</code> and <code>DBMS_LDAP.populate_mod_array()</code> first.
PROCEDURE <code>free_mod_array</code>	Frees the memory allocated by <code>DBMS_LDAP.create_mod_array()</code> .
FUNCTION <code>count_values</code>	Counts the number of values returned by <code>DBMS_LDAP.get_values ()</code> .
FUNCTION <code>count_values_len</code>	Counts the number of values returned by <code>DBMS_LDAP.get_values_len ()</code> .
FUNCTION <code>rename_s</code>	Renames an LDAP entry synchronously.
FUNCTION <code>explode_dn</code>	Breaks a DN up into its components.
FUNCTION <code>open_ssl</code>	Establishes an SSL (Secure Sockets Layer) connection over an existing LDAP connection.

Table 4–1 (Cont.) DBMS_LDAP API Subprograms

Function or Procedure	Description
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.

See Also:

- [Searching by Using DBMS_LDAP](#) for information about the `DBMS_LDAP.search_s()` and `DBMS_LDAP.search_st()` functions
- [Enabling Session Termination by Using DBMS_LDAP](#) for information about the `DBMS_LDAP.unbind_s()` function

Exception Summary

DBMS_LDAP can generate the following exceptions:

Table 4–2 DBMS_LDAP Exception Summary

Exception Name	Oracle Error Number	Cause of Exception
<code>general_error</code>	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.
<code>init_failed</code>	31203	Raised by <code>DBMS_LDAP.init()</code> if there are some problems.
<code>invalid_session</code>	31204	Raised by all functions and procedures in the <code>DBMS_LDAP</code> package if they are passed an invalid session handle.
<code>invalid_auth_method</code>	31205	Raised by <code>DBMS_LDAP.bind_s()</code> if the authentication method requested is not supported.
<code>invalid_search_scope</code>	31206	Raised by all of the 'search' functions if the scope of the search is invalid.

Table 4–2 (Cont.) DBMS_LDAP Exception Summary

Exception Name	Oracle Error Number	Cause of Exception
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.
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.

Table 4–2 (Cont.) DBMS_LDAP Exception Summary

Exception Name	Oracle Error Number	Cause of Exception
invalid_ssl_wallet_loc	31220	Raised by DBMS_LDAP.open_ssl if the wallet location is NULL but the SSL authentication mode requires a valid wallet.
invalid_ssl_wallet_password	31221	Raised by DBMS_LDAP.open_ssl if the wallet password given is NULL.
invalid_ssl_auth_mode	31222	Raised by DBMS_LDAP.open_ssl if the SSL authentication mode is not one of 1, 2 or 3.

Data-Type Summary

The DBMS_LDAP package uses the following data-types:

Table 4–3 *DBMS_LDAP Data-Type Summary*

Data-Type	Purpose
SESSION	Used to hold the handle of the LDAP session. Nearly all of the functions in the API require a valid LDAP session to work.
MESSAGE	Used to hold a handle to the message retrieved from the result set. This is used by all functions that work with 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 4–4 *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 hostname 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 hostname also does not contain the port number, a default port number of 389 is assumed.

Return Values

Table 4–5 *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 4–6 *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 4-7 *SIMPLE_BIND_S Function Parameters*

Parameter	Description
<code>ld</code>	A valid LDAP session handle.
<code>dn</code>	The Distinguished Name of the User that we are trying to login as.
<code>passwd</code>	A text string containing the password.

Return Values

Table 4-8 *SIMPLE_BIND_S Function Return Values*

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

Exceptions

Table 4–9 *SIMPLE_BIND_S Function Exceptions*

Exception	Description
invalid_session	Raised if the session handle <code>ld</code> 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 4–10 BIND_S Function Parameters

Parameter	Description
<code>ld</code>	A valid LDAP session handle
<code>dn</code>	The Distinguished Name of the User that we are trying to login as
<code>cred</code>	A text string containing the credentials used for authentication
<code>meth</code>	The authentication method

Return Values

Table 4–11 BIND_S Function Return Values

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

Exceptions

Table 4–12 *BIND_S Function Exceptions*

Exception	Description
invalid_session	Raised if the session handle <code>ld</code> 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 4–13 UNBIND_S Function Parameters

Parameter	Description
ld	A valid LDAP session handle.

Return Values

Table 4–14 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 4–15 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 4–16 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 4–17 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 4–18 *COMPARE_S Function Exceptions*

Exception	Description
invalid_session	Raised if the session handle <code>ld</code> 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 4–19 SEARCH_S Function Parameters

Parameter	Description
<code>ld</code>	A valid LDAP session handle.
<code>base</code>	The dn of the entry at which to start the search.
<code>scope</code>	One of <code>SCOPE_BASE</code> (0x00), <code>SCOPE_ONELEVEL</code> (0x01), or <code>SCOPE_SUBTREE</code> (0x02), indicating the scope of the search.
<code>filter</code>	A character string representing the search filter. The value <code>NULL</code> can be passed to indicate that the filter "(objectclass=*)" which matches all entries is to be used.
<code>attrs</code>	A collection of strings indicating which attributes to return for each matching entry. Passing <code>NULL</code> for this parameter causes all available user attributes to be retrieved. The special constant string <code>NO_ATTRS</code> ("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 <code>ALL_USER_ATTRS</code> ("*") can be used in the <code>attrs</code> array along with the names of some operational attributes to indicate that all user attributes plus the listed operational attributes are to be returned.
<code>attronly</code>	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 4–19 (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 4–20 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 4–21 SEARCH_S Function Exceptions

Exception	Description
invalid_session	Raised if the session handle <code>ld</code> 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 4–22 SEARCH_ST Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
base	The dn of the entry at which to start the search.
scope	One of SCOPE_BASE (0x00), SCOPE_ONELEVEL (0x01), or SCOPE_SUBTREE (0x02), indicating the scope of the search.
filter	A character string representing the search filter. The value NULL can be passed to indicate that the filter "(objectclass=*)" which matches all entries is to be used.
attrs	A collection of strings indicating which attributes to return for each matching entry. Passing NULL for this parameter causes all available user attributes to be retrieved. The special constant string NO_ATTRS ("1.1") MAY be used as the only string in the array to indicate that no attribute types are to be returned by the server. The special constant string ALL_USER_ATTRS ("*") can be used in the attrs array along with the names of some operational attributes to indicate that all user attributes plus the listed operational attributes are to be returned.

Table 4–22 (Cont.) SEARCH_ST Function Parameters

Parameter	Description
attronly	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 4–23 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 4–24 SEARCH_ST Function Exceptions

Exception	Description
invalid_session	Raised if the session handle <code>ld</code> is invalid.
invalid_search_scope	Raised if the search scope is not one of SCOPE_BASE, SCOPE_ONELEVEL or SCOPE_SUBTREE.
invalid_search_time_value	Raised if the time value specified for the time-out is invalid.
general_error	For all other errors. The error string associated with this exception will explain the error in detail.

Usage Notes

This function is very similar to `DBMS_LDAP.search_s()` except that it requires a time-out value to be given.

See Also

`DBMS_LDAP.search_s()`, `DBML_LDAP.first_entry()`, `DBMS_LDAP.next_entry`.

FUNCTION first_entry

The function first_entry is used to retrieve the first entry in the result set returned by either search_s() or search_st()

Syntax

```
FUNCTION first_entry
(
    ld IN SESSION,
    msg IN MESSAGE
)
RETURN MESSAGE;
```

Parameters

Table 4–25 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 4–26 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 4–27 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 4–28 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 4–29 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 4–30 NEXT_ENTRY Function Exceptions

Exception	Description
invalid_session	Raised if the session handle, <code>ld</code> 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 4–31 *COUNT_ENTRY Function Parameters*

Parameter	Description
<code>ld</code>	A valid LDAP session handle
<code>msg</code>	The search result, as obtained by a call to one of the synchronous search routines

Return Values

Table 4–32 *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 4–33 *COUNT_ENTRY Function Exceptions*

Exception	Description
<code>invalid_session</code>	Raised if the session handle <code>ld</code> is invalid.
<code>invalid_message</code>	Raised if the incoming 'msg' handle is invalid.
<code>count_entry_error</code>	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 4–34 FIRST_ATTRIBUTE Function Parameter**

Parameter	Description
<code>ld</code>	A valid LDAP session handle
<code>ldapentry</code>	The entry whose attributes are to be stepped through, as returned by <code>first_entry()</code> or <code>next_entry()</code>
<code>ber_elem</code>	A handle to a BER ELEMENT that is used to keep track of which attribute in the entry has been read

Return Values**Table 4–35 FIRST_ATTRIBUTE Function Return Values**

Value	Description
<code>VARCHAR2</code> (function return)	The name of the attribute if it exists. NULL if no attribute exists or if an error occurred.
<code>ber_elem</code>	A handle used by <code>DBMS_LDAP.next_attribute()</code> to iterate over all of the attributes

Exceptions**Table 4–36 FIRST_ATTRIBUTE Function Exceptions**

Exception	Description
<code>invalid_session</code>	Raised if the session handle <code>ld</code> is invalid.

Table 4–36 (Cont.) FIRST_ATTRIBUTE Function Exceptions

Exception	Description
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 4–37 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 <code>first_entry()</code> or <code>next_entry()</code> .
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 4–38 NEXT_ATTRIBUTE Function Return Values

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

Exceptions

Table 4–39 NEXT_ATTRIBUTE Function Exceptions

Exception	Description
invalid_session	Raised if the session handle <code>ld</code> 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 4–40 GET_DN Function Parameters**

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

Return Values**Table 4–41 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 4–42 GET_DN Function Exceptions**

Exception	Description
invalid_session	Raised if the session handle <code>ld</code> is invalid.
invalid_message	Raised if the incoming 'msg' handle is invalid.
get_dn_error	Raised if there was a problem in determining the DN

Usage Notes

The function `get_dn()` can be used to retrieve the DN of an entry as the program logic is iterating through the result set. This can in turn be used as an input to `explode_dn()` to retrieve the individual components of the DN.

See Also

`DBMS_LDAP.explode_dn()`.

FUNCTION `get_values`

The function `get_values()` can be used to retrieve all of the values associated 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 4–43 *GET_VALUES Function Parameters*

Parameter	Description
<code>ld</code>	A valid LDAP session handle
<code>ldapentry</code>	A valid handle to an entry returned from a search result
<code>attr</code>	The name of the attribute for which values are being sought

Return Values

Table 4–44 *GET_VALUES Function Return Values*

Value	Description
<code>STRING_COLLECTION</code> (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 4–45 *GET_VALUES Function Exceptions*

Exception	Description
<code>invalid session</code>	Raised if the session handle <code>ld</code> is invalid.
<code>invalid message</code>	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 4–46 *GET_VALUES_LEN Function Parameters*

Parameter	Description
<code>ld</code>	A valid LDAP session handle.
<code>ldapentrymsg</code>	A valid handle to an entry returned from a search result.
<code>attr</code>	The string name of the attribute for which values are being sought.

Return Values

Table 4–47 *GET_VALUES_LEN Function Return Values*

Value	Description
<code>BINVAL_COLLECTION</code> (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 4–48 *GET_VALUES_LEN Function Exceptions*

Exception	Description
invalid_session	Raised if the session handle <code>ld</code> 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 4–49 DELETED_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 4–50 DELETED_S Function Return Values**

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

Exceptions**Table 4–51 DELETED_S Function Exceptions**

Exception	Description
invalid_session	Raised if the session handle ld is invalid.
invalid_entry_dn	Raised if the distinguished name of the entry is invalid
general_error	For all other errors. The error string associated with this exception will explain the error in detail.

Usage Notes

The function `delete_s()` can be used to remove only leaf 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 4–52 MODRDN2_S Function Parameters

Parameter	Description
<code>ld</code>	A valid LDAP session handle.
<code>entrydn</code>	The distinguished name of the entry (This entry must be a leaf node in the DIT).
<code>newrdn</code>	The new relative distinguished name of the entry.
<code>deleteoldrdn</code>	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 4–53 MODRDN2_S Function Return Values

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

Exceptions

Table 4–54 MODRDN2_S Function Exceptions

Exception	Description
invalid_session	Raised if the session handle <code>ld</code> 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 4–55 *ERR2STRING Function Parameters*

Parameter	Description
<code>ldap_err</code>	An error number returned from one the API calls.

Return Values

Table 4–56 *ERR2STRING Function Return Values*

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

Exceptions

Table 4–57 *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 4–58 CREATE_MOD_ARRAY Function Parameters**

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

Return Values**Table 4–59 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 4–60 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 4–61 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 4–62 POPULATE_MOD_ARRAY (String Version) Procedure Return Values

Value	Description
N/A	

Exceptions

Table 4–63 *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 4–64 *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
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

None

Exceptions

Table 4–65 *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 4–66** *MODIFY_S Function Parameters*

Parameter	Description
ld	This parameter is a handle to an LDAP session, as returned by a successful call to <code>DBMS_LDAP.init()</code> .
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 <code>DBMS_LDAP.create_mod_array()</code> .

Return Values**Table 4–67** *MODIFY_S Function Return Values*

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

Exceptions

Table 4–68 *MODIFY_S Function Exceptions*

Exception	Description
invalid_session	Invalid LDAP session
invalid_entry_dn	Invalid LDAP entry dn
invalid_mod_array	Invalid LDAP mod array

Usage Notes

This function call has to follow successful calls of `DBMS_LDAP.create_mod_array()` and `DBMS_LDAP.populate_mod_array()`.

See Also

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

FUNCTION add_s

Adds a new entry to the LDAP directory synchronously. Before calling `add_s`, 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 4–69 ADD_S Function Parameters**

Parameter	Description
ld	This parameter is a handle to an LDAP session, as returned by a successful call to <code>DBMS_LDAP.init()</code> .
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 <code>DBMS_LDAP.create_mod_array()</code> .

Return Values**Table 4–70 ADD_S Function Return Values**

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

Exceptions

Table 4–71 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 4–72 FREE_MOD_ARRAY Procedure Parameters**

Parameter	Description
modptr	This parameter is the handle to an LDAP mod structure, as returned by successful call to <code>DBMS_LDAP.create_mod_array()</code> .

Return Values

None

Exceptions

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 4–73 *COUNT_VALUES Function Parameters*

Parameter	Description
values	The collection of string values.

Return Values

Table 4–74 *COUNT_VALUES Function Return Values*

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

Exceptions

Table 4–75 *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 4–76 COUNT_VALUES_LEN Function Parameters**

Parameter	Description
values	The collection of binary values.

Return Values**Table 4–77 COUNT_VALUES_LEN Function Return Values**

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

Exceptions**Table 4–78 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 4–79 RENAME_S Function Parameters**

Parameter	Description
ld	This parameter is a handle to an LDAP session, as returned by a successful call to <code>DBMS_LDAP.init()</code> .
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 4–80 RENAME_S Function Return Values**

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

Exceptions

Table 4–81 *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 4–82 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 4–83 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 4–84 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 4–85 OPEN_SSL Function Parameters**

Parameter	Description
ld	This parameter is a handle to an LDAP session, as returned by a successful call to <code>DBMS_LDAP.init()</code> .
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 4–86 OPEN_SSL Function Return Values**

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

Exceptions

Table 4–87 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 4–88 MSGFREE Function Parameters

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

Return Values

Table 4–89 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: <ul style="list-style-type: none">■ 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 4–90 *BER_FREE Function Parameters*

Parameter	Description
<code>ber_elem</code>	A handle to BER ELEMENT.
<code>freebuf</code>	The value of this flag should be zero while the BER ELEMENT returned from <code>DBMS_LDAP.first_attribute()</code> 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()`.

Part II

Oracle Extensions to LDAP APIs

Part II explains Oracle-specific APIs, and includes some basic Oracle9i Database Server Release 2 (9.2) concepts of users and groups, Java API information (`oracle.ldap.util` classes), PL/SQL API information, and information about provisioning interfaces. It contains these chapters:

- [Chapter 5, "Overview of Oracle Extensions"](#)
- [Chapter 6, "Java API for Oracle Internet Directory"](#)
- [Chapter 7, "The DBMS_LDAP_UTL PL/SQL Package"](#)
- [Chapter 8, "Developing Provisioning-Integrated Applications"](#)
- [Chapter 9, "Oracle Internet Directory Server Plug-in Framework"](#)

Overview of Oracle Extensions

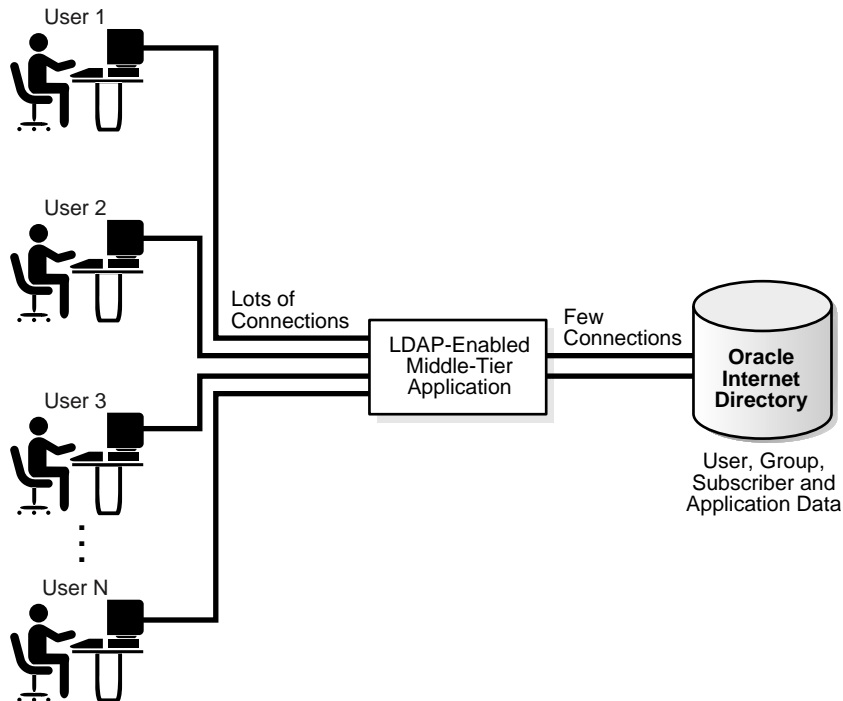
This chapter explains how to directory-enable your applications. It contains these topics:

- [The LDAP Access Model](#)
- [Entities Modeled in LDAP](#)
- [API Enhancements: Overview & Usage Model](#)
- [API Enhancements: Overview & Usage Model](#)
- [API Enhancements: Assumptions](#)
- [Installation and First Use](#)

The LDAP Access Model

Most directory-enabled applications are middle-tiers that handle multiple user requests simultaneously. [Figure 5-1](#) shows the usage of LDAP in such middle-tier environments.

Figure 5-1 Oracle Internet Directory in a Middle-Tier Environment



As [Figure 5-1](#) shows, applications act as middle-tiers, or backend programs, that multiple users can access. If a user request needs an LDAP operation to be performed, then these applications perform the operation by using a smaller set of pre-created connections to Oracle Internet Directory.

This section tells you how to implement this access model. It contains these sections:

- [Application Installation Logic](#)
- [Application Startup and Bootstrap Logic](#)

- [Application Runtime Logic](#)
- [Application Shutdown Logic](#)
- [Application Deinstallation Logic](#)

Application Installation Logic

1. Create in Oracle Internet Directory an identity corresponding 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 have to be performed on behalf of the user

Application Startup and Bootstrap Logic

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 should retrieve that metadata and initialize other parts of the application.

The application should then establish a pool of connections to serve user requests.

Application Runtime Logic

For every end-user request that needs an LDAP operation, the application should:

- Pick a connection from the pool of LDAP connections
- If Oracle9iAS Single Sign-On is not used, then authenticate the end-user if required
- If the LDAP operation needs to be performed with the effective rights of the end-user, then switch the user to the end-user identity
- Perform the LDAP operation by using regular API or the enhancements to it described in this chapter
- Once the operation is complete, if the application performed a proxy operation, then ensure that the effective user is now the application identity itself

- Return the LDAP connection back to the pool of connections

Application Shutdown Logic

Abandon any outstanding LDAP operations and close all LDAP connections.

Application Deinstallation Logic

Remove the application identity and the associated LDAP authorizations granted to the application identity.

Entities Modeled in LDAP

Oracle enhancements to the LDAP API help applications get or set LDAP information for these entities:

Entity	Description
Users	Enterprise users represented in Oracle Internet Directory who have access to one or more applications.
Groups	Aggregations of enterprise users that typically signify an authorization. Directory-enabled applications that store these aggregations in the directory must be able to locate groups and query group membership.
Subscribers	Entities modeled in the hosted environment. A subscriber is typically a company that subscribes to one or more Oracle hosting-enabled products.

The rest of this section describes what applications need from Oracle Internet Directory for these entities. It contains these topics:

- [Users](#)
- [Groups](#)
- [Subscribers](#)

Users

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

Groups

Groups are modeled in Oracle Internet Directory as a collection of distinguished names. Directory-enabled applications need to access Oracle Internet Directory for the following group related operations 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

Subscribers

Subscribers are entities or organizations that subscribe to the hosting-enabled services offered in the Oracle product stack. Directory-enabled applications need to

access Oracle Internet Directory to get subscriber properties—for example, user search base or password policy—and to create a new subscriber.

A subscriber is typically identified by one of the following:

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

API Enhancements: Overview & Usage Model

As described in the preceding section, there are several conventions that all applications integrating with the directory need to follow. The primary goal of the API enhancements described in this chapter help you conform your applications to these conventions.

API Enhancements: Assumptions

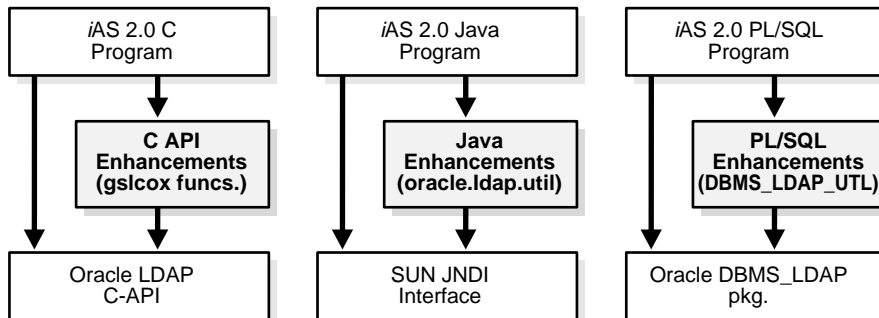
The API enhancements described in this chapter rest on the following assumptions:

- All of these API functions that require an LDAP connection assume that the application has already established it by using the appropriate mechanisms in the respective programming language
- The LDAP connection being passed into these API functions is associated with an identity that has the necessary permissions to perform the given operation. In other words, the new API functions do not have their own authorization model. Rather, they rely on the LDAP authorization model. If a certain operation fails because of insufficient authorization, the error is simply transmitted back to the invoking application.
- All Java-based applications use Sun's JNDI as the interface for LDAP connectivity
- All PL/SQL based applications use Oracle's `DBMS_LDAP` package for LDAP connectivity

System Placement

Figure 5–2 shows the placement of the API enhancements in relation to existing APIs:

Figure 5–2 Placement of Oracle API Enhancements



As Figure 5–2 shows, in the languages—PL/SQL and Java—the API enhancements described in this chapter 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

Applications need to access the underlying APIs for such common things as connection establishment and connection closing. They can also use the existing APIs to look up other LDAP entries not covered by the API enhancements.

API Enhancements Functional Categorization

Based on the entities on which they operate, these API enhancements can be categorized as follows:

- User Management—This functionality allows applications to get or set various user related properties
- Group Management—This functionality allows applications to query group properties
- Subscriber Management—This functionality allows applications to get or set such subscriber-related properties as user search base

- Application Management—This functionality allows applications to manage certain application metadata in Oracle Internet Directory
- Miscellaneous—This functionality—generate GUID—is universally applicable

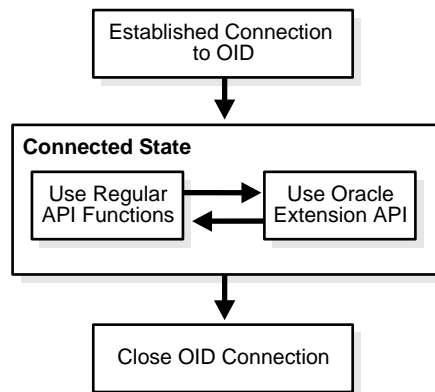
API Enhancements Usage Model

The primary users of the enhancements described in this chapter are middle-tier or back-end applications that must perform LDAP lookups for users, groups, applications, or subscribers. This section describes how these applications integrate these API enhancements into their logic—that is, the usage of the API enhancements only.

See Also: ["The LDAP Access Model"](#) on page 5-2 for a conceptual description of the usage model

[Figure 5-3](#) shows the programmatic flow of control for using the API enhancements described in this chapter:

Figure 5-3 Programmatic Flow of API Enhancements



As [Figure 5-3](#) shows, the applications first establish a connection to Oracle Internet Directory. They can then use existing API functions and the API enhancements interchangeably.

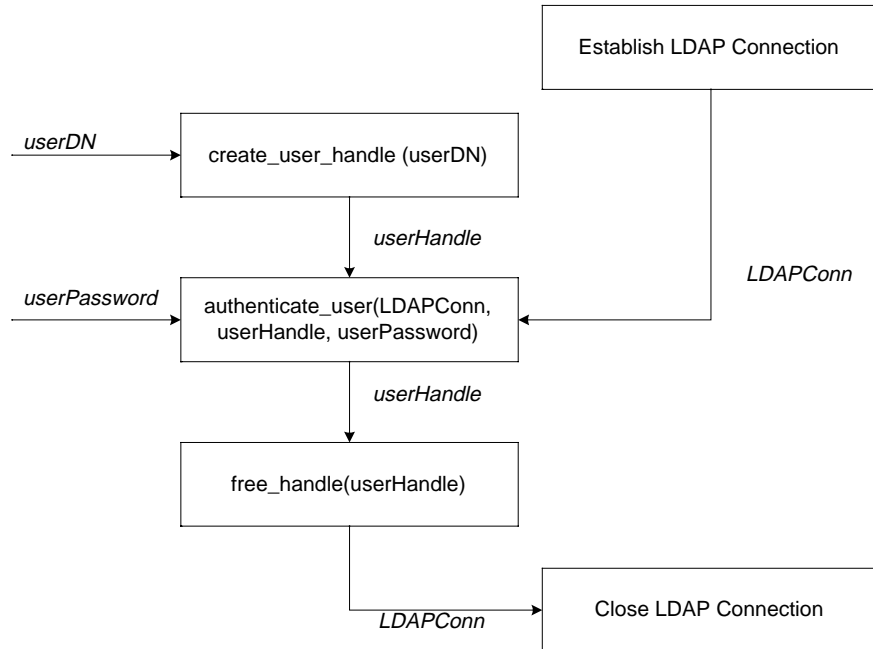
Programming Abstractions for the PL/SQL Language

Most of the enhancements described in this chapter provide helper functions to access data in relation to such specific LDAP entities as users, groups, subscribers and applications. In many cases, you have to pass a reference to one of these entities to the API functions. These API enhancements 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 Oracle9iAS 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 5–4 illustrates this usage model.

Figure 5–4 Programming Abstractions for the PL/SQL Language



Programming Abstractions for the Java Language

Instead of handles, LDAP entities—that is, users, groups, subscribers, 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, an application that needs to authenticate a user would have to 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.

User Management Functionality

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

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

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.

Installation and First Use

The Java API is installed as part of the LDAP client installation.

The PL/SQL API are installed as part of the Oracle9i Database installation. To use the PL/SQL API, you must load it by using a script, called `catldap.sql`, located in `$ORACLE_HOME/rdbms/admin`.

Java API for Oracle Internet Directory

This chapter contains reference material for the Java API for Oracle Internet Directory.

This chapter contains these sections:

- [Class Descriptions](#)
- [Classes](#)
- [Exceptions](#)

Class Descriptions

This section describes classes. It contains these topics:

- [User Class](#)
- [Subscriber Class](#)
- [Group Class](#)
- [PropertySetCollection, PropertySet, and Property Classes](#)

User Class

The user class is used to represent a particular user under a subscriber. You can create a user object using a DN, a GUID, or a simple name, along with the appropriate subscriber identification also based on a DN, a GUID, or a simple name. When a simple name is used, additional information from the Root Oracle Context and the Subscriber Oracle Context is used to identify the user. An example of a user construction follows:

```
User myuser = new User ( ctx,          // A valid InitialDirContext
                        Util.IDTYPE_DN,
                        "cn=user1,cn=users,o=oracle,dc=com" ,
                        Util.IDTYPE_DN,
                        "o=oracle,dc=com" ,
                        false);
```

`myuser` is defined in the previous example using its DN. The corresponding subscriber is also identified by its DN. If you have an existing subscriber object, you can also create a user object directly by using the subscriber object. For example, given a subscriber object, `myOracleSubscriber`, representing "o=oracle,dc=com", you can create the same user object as above by using the following:

```
User myuser = new User ( ctx,          // A valid InitialDirContext
                        Util.IDTYPE_DN,
                        "cn=user1,cn=users,o=oracle,dc=com" ,
                        myOracleSubscriber,
                        false);
```

Some common user object uses include setting and getting user properties, and authenticating the user. An example of authenticating a user follows:

```
if(myuser.authenticateUser( ctx
                        User.CREDTYPE_PASSWD,
                        "welcome" )) {
    // do work here
}
```

In the previous example, the user is authenticated using the clear text password "welcome".

The following is an example of getting the telephone number of the user:

```
String[] userAttrList = {"telephonenumber"};
PropertySetCollection result = myuser.getProperties( ctx,userAttrList );
```

See Also: ["Java Sample Code"](#) on page B-34 for more sample uses of the user class

Subscriber Class

The subscriber class is used to represent a subscriber with a valid Oracle Context. You can create a subscriber object using a DN, a GUID, or a simple name. When a simple name is used, additional information from the Root Oracle Context is used to identify the user. A default subscriber object creation is also supported. The information regarding the default subscriber is stored in the Root Oracle Context. An example of a subscriber construction follows:

```
Subscriber mysub = new Subscriber( ctx, //a valid InitialDirContext
                                Util.IDTYPE_DN,
                                "o=oracle,dc=com",
                                false );
```

`mysub` is defined in the previous example by its DN, "o=oracle,dc=com".

A common subscriber object use is getting subscriber properties is. For example:

```
String[] attrList = { "cn", "orclguid" };
PropertySetCollection result= mysub.getProperties(ctx,attrList);
// do work with result
```

A subscriber object can also be used during a user object construction to identify the subscriber. An example to create a user object with simple name "user1" under the subscriber created above follows:

```
myuser1 = new User ( ctx, //a valid InitialDirContext
                    Util.IDTYPE_SIMPLE,
                    "user1",
                    mysub,
                    false );
```

See Also: ["Java Sample Code"](#) on page B-34 for more sample uses of the subscriber class

Group Class

The group is used to represent a valid group entry. You can create a group object using its DN or GUID. An example of a group construction follows:

```
Group mygroup = new Group ( Util.IDTYPE_DN,
                            "cn=group1,cn=Groups,o=oracle,dc=com" );
```

`mygroup` is defined in the previous example by its DN.

A sample usage of the group object is getting group properties. For example:

```
PropertySetCollection result = mygroup.getProperties( ctx, null );
```

The group class also supports membership related functionality. Given a user object, you can find out if it is a direct or a nested member of a group by using the `ismember()` method. For example:

```
if (mygroup.isMember( ctx, // a valid InitialDirContext
                    myuser,
                    true ) ) { // set to true for nested member
    // do work
}
```

`myuser` is a user object. The third argument is set to `true` to indicate that nested membership is being considered. If set to `false`, then only direct membership is considered.

You can also obtain a list of groups that a particular user belongs to using `Util.getGroupMembership()`. For example:

```
PropertySetCollection result = Util.getGroupMembership( ctx,
                                                    myuser,
                                                    new String[0],
                                                    true );
```

See Also: ["Java Sample Code"](#) on page B-34 for more sample uses of the group class

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`, identified by a DN. A `PropertySet` can contain attribute(s), 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));
    }
}
```

`myuser` is a user object. `psc` 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-34 for more sample uses of the PropertySetCollection, PropertySet, and Property classes

Classes

This section explains classes.

This section contains these topics:

- [oracle.ldap.util.Base64](#)
- [oracle.ldap.util.Group](#)
- [oracle.ldap.util.Guid](#)
- [oracle.ldap.util.LDIF](#)
- [oracle.ldap.util.LDIFAttribute](#)
- [oracle.ldap.util.LDIFAttribute](#)
- [oracle.ldap.util.LDIFMigration](#)
- [oracle.ldap.util.LDIFReader](#)
- [oracle.ldap.util.LDIFRecord](#)
- [oracle.ldap.util.LDIFSubstitute](#)
- [oracle.ldap.util.LDIFWriter](#)
- [oracle.ldap.util.Property](#)
- [oracle.ldap.util.PropertySet](#)
- [oracle.ldap.util.PropertySetCollection](#)
- [oracle.ldap.util.Subscriber](#)
- [oracle.ldap.util.User](#)
- [oracle.ldap.util.Util](#)

oracle.ldap.util.Base64

Syntax:

```
public class oracle.ldap.util.Base64
```

Description:

Provides encoding of raw bytes to base64-encoded bytes and decoding of base64-encoded bytes to raw bytes.

Constructors

Base64()**Syntax:**

```
public Base64()
```

Methods

encode(String inStr)**Syntax:**

```
public static java.lang.String encode(String inStr)
```

Description:

Use this method to convert a string to base64-encoded string.

Parameters:

inStr - String to be base64 encoded.

Returns:

outStr Base64 encoded string.

decode(String inStr)**Syntax:**

```
public static java.lang.String encode(String inStr)
```

Description:

Use this method to convert a string to base64-encoded string.

Parameters:

inStr - String to be base64 encoded.

Returns:

outStr Base64 encoded string.

decode(String inStr)

Syntax:

```
public static java.lang.String decode(String inStr)
```

Description:

Use this method to decode a base64-encoded string to the original string.

Parameters:

inStr - Base64 encoded string

Returns:

outStr - The original string.

encode(byte[] inBytes)

Syntax:

```
public static byte[] encode(byte[] inBytes)
```

Description:

Returns an array of base64-encoded characters to represent the passed data array.

Parameters:

data - the array of bytes to encode

Returns:

Base64-coded byte array.

decode(byte[] inBytes)

Syntax:

```
public static byte[] decode(byte[] inBytes)
```

Description:

Decode a base64-encoded sequence bytes. All illegal symbols in the input are ignored (CRLF, Space).

Parameters:

inpBytes - A sequence of base64-encoded bytes

Returns:

The original data from the base64 input.

oracle.ldap.util.Group

Syntax:

```
public class oracle.ldap.util.Group
```

Constructors

Group(int inGroupIdType, String inGroupName)

Syntax:

```
public Group(int inGroupIdType, String inGroupName)
```

Description:

Constructs a group using a group ID along with its type.

Parameters:

inGroupIdType - The type of group ID being used - either Util.IDTYPE_DN or Util.IDTYPE_GUID

inGroupName - the group ID

Methods

oracle.ldap.util.PropertySetCollection `getProperties(DirContext ctx, String[] attrList)`.

Syntax:

```
public oracle.ldap.util.PropertySetCollection getProperties(DirContext ctx,  
String[] attrList)
```

Description:

Retrieves selected attributes associated with this group.

Parameters:

`ctx` - A valid `DirContext`

`attrList` - An array of attributes to be retrieved. Null indicates that all attributes should be retrieved. An empty array indicates that none should be retrieved.

Returns:

A `PropertySetCollection` of the results.

resolve(DirContext ctx)

Syntax:

```
public void resolve(DirContext ctx)
```

Description:

Validates the group by identifying its DN.

Parameters:

`ctx` - A valid `DirContext`

getDn(DirContext ctx)

Syntax:

```
public java.lang.String getDn(DirContext ctx)
```

Description:

Returns the DN of this group.

Returns:

The DN of this group.

boolean isMember(DirContext ctx, User user, boolean nested)

Syntax:

```
public boolean isMember(DirContext ctx, User user, boolean nested)
```

Description:

Checks if a particular user is a member of this group.

Parameters:

ctx - A valid DirContext

group - A valid User object

nested - Set to true if nested memberships are allowed. Otherwise, only direct memberships are considered.

Returns:

True if the given user is a member of this group, false otherwise.

oracle.ldap.util.Guid

Syntax:

```
public final class oracle.ldap.util.Guid implements java.lang.Cloneable
```

Description:

This class represents GUIDs (Globally Unique Identifiers), or object IDs. This is an immutable class.

Constructors

Guid()

Syntax:

```
public Guid()
```

Description:

Default constructor.

Guid(String guid)

Syntax:

```
Guid(String guid)
```

Description:

Constructs a GUID from a string.

Parameters:

guid - a string representation of a Globally Unique Identifier

Guid(byte[] byte_array)

Syntax:

```
public Guid(byte[] byte_array)
```

Description:

Constructs a GUID from a byte array.

Parameters:

byte_array - an array of bytes representing a GUID. This constructor will validate the length of the byte array before proceeding.

Methods

newInstance()

Syntax:

```
public static oracle.ldap.util.Guid newInstance()
```

Description:

generates a new GUID.

Returns:

A new instance of the GUID class.

getBytes()

Syntax:

```
public byte[] getBytes()
```


Description:

Return the 'byte' form of the GUID.

Returns:

Return the 'byte' form of the GUID.

toString()**Syntax:**

```
public final java.lang.String toString()
```

Description:

Retrieves the GUID in a string format.

Returns:

The GUID in a string format.

equals (Object o)**Syntax:**

```
public boolean equals(Object o)
```

Description:

Compares the GUID in a string format.

Returns:

True if they are equal, false otherwise.

hashCode()**Syntax:**

```
public int hashCode()
```

Description:

Returns the hashcode of this object for hashing purposes.

Returns:

The integer hashcode of the GUID.

Object clone()

Syntax:

```
public java.lang.Object clone()
```

Description:

Clones a GUID object.

Returns:

A clone of an existing GUID object.

Fields

GUID_BYTE_SIZE

Syntax:

```
public static final GUID_BYTE_SIZE
```

Description:

Number of bytes required for GUID.

GUID_STRING_SIZE

Syntax:

```
public static final GUID_STRING_SIZE
```

Description:

The number of bytes required for the string representation of GUID.

oracle.ldap.util.LDIF

Syntax:

```
public class oracle.ldap.util.LDIF
```

Description:

A class that defines most common thing that pertains to LDAP data interchange format.

Fields

RECORD_CHANGE_TYPE_ADD

Syntax:

```
static final RECORD_CHANGE_TYPE_ADD
```

Description:

Record change type - Add

RECORD_CHANGE_TYPE_DELETE

Syntax:

```
static final RECORD_CHANGE_TYPE_DELETE
```

Description:

Record change type - Delete

RECORD_CHANGE_TYPE_MODIFY

Syntax:

```
static final RECORD_CHANGE_TYPE_MODIFY
```

Description:

Record change type - Modify

RECORD_CHANGE_TYPE_MODDN

Syntax:

```
static final RECORD_CHANGE_TYPE_MODDN
```

Description:

Record change type - MODDN

ATTRIBUTE_CHANGE_TYPE_ADD

Syntax:

```
static final ATTRIBUTE_CHANGE_TYPE_ADD
```

Description:

Attribute change type - Add

ATTRIBUTE_CHANGE_TYPE_DELETE

Syntax:

```
static final ATTRIBUTE_CHANGE_TYPE_DELETE
```

Description:

Attribute change type - Delete

ATTRIBUTE_CHANGE_TYPE_REPLACE

Syntax:

```
static final ATTRIBUTE_CHANGE_TYPE_REPLACE
```

Description:

Attribute change type - Replace

oracle.ldap.util.LDIFAttribute

Syntax:

```
public class oracle.ldap.util.LDIFAttribute
```

Description:

The LDIFAttribute class represents the name and values of an attribute. It is used to specify an attribute to be added to, deleted from, or modified in a directory entry. It is also returned on a search of a directory.

Constructors

LDIFAttribute(String attrName)

Syntax:

```
public LDIFAttribute(String attrName)
```

Description:

Constructs an attribute with no values.

Parameters:

attrName - Name of the attribute

LDIFAttribute(LDIFAttribute ldapAttribute)

Syntax:

```
public LDIFAttribute(LDIFAttribute ldapAttribute)
```

Description:

Constructs an attribute with copies of all values of the input `LDIFAttribute`.

Parameters:

ldapAttribute - An attribute to use as template.

LDIFAttribute(String attrName, byte[] attrBytes)

Syntax:

```
public LDIFAttribute(String attrName, byte[] attrBytes)
```

Description:

Constructs an attribute with a byte-formatted value.

Parameters:

attrName - Name of the attribute

attrBytes - Value of the attribute as raw bytes

LDIFAttribute(String attrName, String attrString)

Syntax:

```
public LDIFAttribute(String attrName, String attrString)
```

Description:

Constructs an attribute that has a single string value.

Parameters:

attrName - name of the attribute

attrString - value of the attribute in String format

LDIFAttribute(String attrName, String[] attrStrings)

Syntax:

```
public LDIFAttribute(String attrName, String[] attrStrings)
```

Description:

Constructs an attribute that has an array of string values.

Parameters:

attrName - name of the attribute

attrStrings - the list of string values for this attribute

Methods

addValue(String attrString)

Syntax:

```
public void addValue(String attrString)
```

Description:

Adds a string value to the attribute.

Parameters:

attrString - Value of the attribute as a string

addValue(byte[] attrBytes)

Syntax:

```
public synchronized void addValue(byte[] attrBytes)
```

Description:

Adds a byte-formatted value to the attribute.

Parameters:

attrBytes - the value of attribute as raw bytes is added to the attribute

addValue(String[] attrValues)**Syntax:**

```
public synchronized void addValue(String[] attrValues)
```

Description:

Adds an array of string values to the attribute.

Parameters:

values - array of string values, add to the attribute

getBytesValues()**Syntax:**

```
public java.util.Enumeration getBytesValues()
```

Description:

Returns an enumerator for the values of the attribute in byte[] format.

Parameters:

A set of attribute values. Each element in the enumeration is of type byte.

getStringValues()**Syntax:**

```
public java.util.Enumeration getStringValues()
```

Description:

Returns an enumerator for the string values of an attribute.

Returns:

An enumerator for the string values.

getBytesValueArray()**Syntax:**

```
public byte[][] getBytesValueArray()
```

Description:

Returns the values of the attribute as an array of byte[].

Returns:

Array of attribute values in byte format.

getStringValueArray()

Syntax:

```
public java.lang.String[] getStringValueArray()
```

Description:

Returns the values of the attribute as an array of strings.

Returns:

Array of attribute values as string object.

setValues(String[] attrValues)

Syntax:

```
public void setValues(String[] attrValues)
```

Description:

Sets the string values as the attribute's values.

Parameters:

attrValues - An array of string values which represent the attribute values.

getLangSubtype()

Syntax:

```
public java.lang.String getLangSubtype()
```

Description:

Returns the language subtype if any. For example, if the attribute name is `cn;lang-fr;phonetic`, this method returns the string `lang-fr`.

Returns:

The language subtype, or null (if the name has no language subtype).

getBaseName(String attrName)**Syntax:**

```
public static java.lang.String getBaseName(String attrName)
```

Description:

Returns the base name. For example, if the attribute name is `cn;lang-fr;phonetic`, then this method returns `cn`.

Parameters:

`attrName` - Name of the attribute to extract the base name from.

Returns:

Base name (for example, the attribute name without subtypes).

getBaseName()**Syntax:**

```
public java.lang.String getBaseName()
```

Description:

Returns the base name of this object. For example, if the attribute name is `cn;lang-fr;phonetic`, then this method returns `cn`.

Returns:

Base name (for example, the attribute name without subtypes).

getName()**Syntax:**

```
public java.lang.String getName()
```

Parameters:

Returns the name of the attribute.

Returns:

Attribute name.

getSubtypes(String attrName)**Syntax:**

```
public static java.lang.String[] getSubtypes(String attrName)
```

Description:

Extracts the subtypes from the specified attribute name. For example, if the attribute name is `cn;lang-fr;phonetic`, then this method returns an array containing `lang-fr` and `phonetic`.

Parameters:

`attrName` - Name of the attribute to extract the subtypes from.

Returns:

Array of subtypes, or null (if no subtypes).

getSubtypes()**Syntax:**

```
public java.lang.String[] getSubtypes()
```

Description:

Extracts the subtypes from the attribute name of this object. For example, if the attribute name is `cn;lang-fr;phonetic`, then this method returns an array containing `lang-ja` and `phonetic`.

Returns:

Array of subtypes, or null (if no subtypes).

hasSubtype(String subtype)**Syntax:**

```
public boolean hasSubtype(String subtype)
```

Description:

Reports whether the attribute name contains the specified subtype. For example, if you check for the subtype `lang-fr` and the attribute name is `cn;lang-fr`, then this method returns `true`.

Parameters:

subtype - The single subtype to check for

Returns:

True if the attribute name contains the specified subtype.

hasSubtypes(String[] subtypes)**Syntax:**

```
public boolean hasSubtypes(String[] subtypes)
```

Description:

Reports if the attribute name contains all specified subtypes. For example, if you check for the subtypes `lang-fr` and `phonetic` and if the attribute name is `cn;lang-fr;phonetic`, then this method returns `true`. If the attribute name is `cn;phonetic` or `cn;lang-fr`, then this method returns `false`.

Parameters:

subtypes - An array of subtypes to check for

Returns:

True if the attribute name contains all subtypes.

removeValue(String attrString)**Syntax:**

```
public synchronized void removeValue(String attrString)
```

Description:

Removes a string value from the attribute.

Parameters:

attrString - The string value to remove

removeValue(byte[] attrBytes)**Syntax:**

```
public void removeValue(byte[] attrBytes)
```

Description:

Removes a byte-formatted value from the attribute.

Parameters:

attrBytes - A byte formatted value to remove

size()

Syntax:

```
public int size()
```

Description:

Returns the number of values of the attribute.

Returns:

Number of values for this attribute.

getChangeType()

Syntax:

```
public int getChangeType()
```

Description:

Return this the change type associated with this attribute (if any).

Returns:

A Change Type constant defined in the LDIF class.

setChangeType(int changeType)

Syntax:

```
public void setChangeType(int changeType)
```

Description:

Sets the change type for this attribute.

Parameters:

changeType - Change type constant defined in the LDIF class.

getValue()**Syntax:**

```
public java.lang.String getValue()
```

Description:

Returns the value of a single value attribute. In case of a multivalued attribute the first value is returned. If the attribute does not contain any value then null is returned.

Returns:

An attribute value or null if the attribute has no value.

contains(String attrString)**Syntax:**

```
public boolean contains(String attrString)
```

Description:

Reports whether this object contains the specified attribute value.

Parameters:

attrString - The value as string object that needs to be checked for

Returns:

True if the attribute contains the specified value, else false.

contains(byte[] attrBytes)**Syntax:**

```
public boolean contains(byte[] attrBytes)
```

Description:

Reports whether this object contains the specified attribute value.

Parameters:

attrValue - The value as byte-formatted representation that needs to be checked for

Returns:

True if the attribute contains the specified value, else false.

toString()**Syntax:**

```
public java.lang.String toString()
```

Description:

Retrieves the string representation of an attribute in an LDAP entry. For example:
LDIFAttribute type='cn', values='Barbara Jensen,Babs Jensen'

Returns:

String representation of the attribute.

equals(Object ldifAttr)**Syntax:**

```
public boolean equals(Object ldifAttr)
```

oracle.ldap.util.LDIFMigration

Syntax:

```
public class oracle.ldap.util.LDIFMigration
```

Description:

This class provides methods to migrate the user information present in the component specific repositories to OID. The input to this migration process is an intermediate LDIF file which contains the substitution variables that needs to be replaced. The output of this process is an LDIF file which could be used to upload the data using any one of the existing tools.

Constructors**LDIFMigration(String inputFile, Vector subsVect, String outFile)****Syntax:**

```
public LDIFMigration(String inputFile, Vector subsVect, String outFile)
```

Description:

This method constructs an object to read the LDIF file, does the substitution, and writes the LDIF entries to a file.

Parameters:

inputFile - Name of the input file

subsVect - The vector containing the substitution variables and the values alternatively

outFile - Name of the output file

Throws:

MigrationException - A migration error could occur due to an I/O error or invalid input parameters. The error code and the the error message of this exception object describes the contexts.

LDIFMigration(File inpF, Vector subsVect, File outF)

Syntax:

```
public LDIFMigration(File inpF, Vector subsVect, File outF)
```

Description:

This method constructs an object to read the LDIF entry from the specified file object, does the substitution, and writes the LDIF entries to the specified file object.

Parameters:

inpF - The input file object

subsVect - The vector containing the substitution variables and the values alternatively

outF - The output file object

Throws:

MigrationException - A migration error could occur due to an I/O error or invalid input parameters. The error code and the the error message of this exception object describes the contexts.

LDIFMigration(InputStream inpS, Vector subsVect, OutputStream outS)

Syntax:

```
public LDIFMigration(InputStream inpS, Vector subsVect, OutputStream outS)
```

Description:

This method constructs an object to read the LDIF entries from the specified input stream, does the substitution, and writes the LDIF entries to the specified output stream.

Parameters:

inpS - The input stream from which provides the LDIF entries

subsVect - The vector containing the substitution variables and the values alternatively

outS - The output stream to which the LDIF entries are written

Throws:

MigrationException - A migration error could occur due to an I/O error or invalid input parameters. The error code and the the error message of this exception object describes the contexts.

Methods

setDirContext(DirContext dirCtx)

Syntax:

```
public void setDirContext(DirContext dirCtx)
```

Description:

Sets the Directory Context.

Parameters:

dirCtx - The directory context object from which the directory attributes are queried to automatically find out the substitution variables.

setUserDN(String dn)

Syntax:

```
public void setUserDN(String dn)
```

Description:

Sets current user dn.

Parameters:

dn - The dn of the user binding

int migrate()

Syntax:

```
public int migrate()
```

Description:

Call this method to read the intermediate LDIF file, do the substitution, and write a new LDIF output file.

Returns:

int number of entries successfully written

Throws:

MigrationException - If an error occurs while reading from or writing to an LDIF file

int migrate(Subscriber subscriber)

Syntax:

```
public int migrate(Subscriber subscriber)
```

Description:

Call this method to read the intermediate LDIF file, do the substitution, and write a new LDIF output file. The substitution variables will be automatically figured out by connecting to the directory server. Following is the list of substitution variables which will be determined from a given LDAP host. s_SubscriberDN, s_UserContainerDN, s_GroupContainerDN, s_SubscriberOracleContextDN, s_RootOracleContextDN

Parameters:

subscriber - The subscriber for which the substitution variables needs to be figured out

Returns:

int number of entries successfully written.

Throws:

MigrationException - If an error occurs while reading from or writing to an LDIF file or a NamingException occurs while performing a directory operation.

cleanup()**Syntax:**

```
public void cleanup()
```

Description:

Closes the LDIF reader and writer streams.

Throws:

MigrationException - if an I/O error occurs while closing the reader or writer streams.

oracle.ldap.util.LDIFReader**Syntax:**

```
public class oracle.ldap.util.LDIFReader
```

Description:

LDAP Data Interchange Format (LDIF) is a file format used to represent the Directory entries. The data from the directory can be exported in this format and could be imported into another directory server. LDIF Data can describe a set of changes that needs to be applied to data in a directory. This format is described in the Internet draft, The LDAP Data Interchange Format (LDIF) - RFC-2849.

This class provides a set of methods to read and manipulate an LDIF file.

Constructors**LDIFReader()****Syntax:**

```
public LDIFReader()
```

Description:

Default Constructor, reads the data from standard input. The input data is required to be present in UTF8 format.

Throws:

IOException - If an I/O error occurs

LDIFReader(String file)**Syntax:**

```
public LDIFReader(String file)
```

Description:

Constructs an input stream reader to read the LDIF data from the file specified.

Parameters:

file - The name of the LDIF file

Throws:

IOException - If an I/O error occurs

LDIFReader(File fileObj)**Syntax:**

```
public LDIFReader(File fileObj)
```

Description:

Constructs an input stream reader to read the LDIF data from the file object specified.

Parameters:

fileObj - The file object of the LDIF file

Throws:

IOException - If an I/O error occurs

LDIFReader(InputStream ds)**Syntax:**

```
public LDIFReader(InputStream ds)
```

Description:

Constructs an input stream reader to read the LDIF data from the specified input stream.

Parameters:

dst - The input stream providing the LDIF data

Throws:

IOException - If an I/O error occurs

Methods

nextEntry()

Syntax:

```
public java.util.Vector nextEntry()
```

Description:

Returns the next entry in the LDIF file. Using this method you can iterate through all the entries in the LDIF file.

Returns:

The next entry as a vector object containing the attributes as name:value pairs, for example, each element in the vector will look like name:value. A null is returned if there are no more entries.

Throws:

IOException - If an I/O error occurs

nextRecord()

Syntax:

```
public synchronized oracle.ldap.util.LDIFRecord nextRecord()
```

Description:

Returns the next record in the LDIF file. Using this method you can iterate through all the entries in the LDIF file.

Returns:

The next entry as a LDIFRecord object. A null is returned if there are no more entries.

Throws:

IOException - If an I/O error occurs

close()**Syntax:**

```
public void close()
```

Description:

Closes the stream.

Throws:

IOException - If an error occurs

oracle.ldap.util.LDIFRecord

Syntax:

```
public class oracle.ldap.util.LDIFRecord implements java.lang.Cloneable
```

Description:

LDIFRecord represents a single entry in an LDIF file, consisting of a distinguished name (DN) and zero or more attributes.

Parameters:

dn - The distinguished name of the new entry

Constructors**LDIFRecord()****Syntax:**

```
public LDIFRecord()
```

Description:

Constructs an LDIFRecord object without dn and attribute values set.

LDIFRecord(String dn)

Syntax:

```
public LDIFRecord(String dn)
```

Description:

Constructs a record with the specified dn.

Parameters:

dn - The distinguished name of the new entry

addAttribute(LDIFAttribute atr)

Syntax:

```
public void addAttribute(LDIFAttribute atr)
```

Description:

Adds an attribute to this record.

Parameters:

atr - The LDIFAttribute object which is to be added

getDN()

Syntax:

```
public java.lang.String getDN()
```

Description:

Returns the distinguished name of the current record.

Returns:

The distinguished name of the current record.

setDN(String dn)

Syntax:

```
public void setDN(String dn)
```

Description:

Sets the dn of this record.

Parameters:

dn - The distinguished name that will be set in the current record

synchronized oracle.ldap.util.LDIFAttribute getAttribute(String attrName)

Syntax:

```
public synchronized oracle.ldap.util.LDIFAttribute getAttribute(String attrName)
```

Description:

Returns the LDIFAttribute object of the specified attribute name.

Parameters:

attrName - Name of the attribute

java.util.Enumeration getAll()

Syntax:

```
public java.util.Enumeration getAll()
```

Description:

Returns an enumeration of the attributes in this record.

Returns:

An enumeration containing all the LDIFAttribute objects.

synchronized java.util.Enumeration getIDs()

Syntax:

```
public synchronized java.util.Enumeration getIDs()
```

Description:

Retrieves an enumeration of the IDs of the attributes in this record as string objects.

Returns:

A non-null enumeration of the attributes' IDs in this record set. If attribute set has zero attributes, then an empty enumeration is returned.

int getChangeType()

Syntax:

```
public int getChangeType()
```

Description:

Retrieves the change type of this record. Change type constants are defined in the LDIF class.

int size()

Syntax:

```
public int size()
```

Description:

Returns the number of attributes in this record. The count does not include the dn.

Returns:

The number of attributes in this record.

Object clone()

Syntax:

```
public java.lang.Object clone()
```

Description:

Creates a replica of this object.

Returns:

A replica of this object.

toString()

Syntax:

```
public java.lang.String toString()
```

Description:

A string representation of this object.

oracle.ldap.util.LDIFSubstitute

Syntax:

```
public class oracle.ldap.util.LDIFSubstitute
```

Description:

This class provides some general methods to do the substitution in an LDIF entry. The LDIF entry is represented in a vector object. The substitution variables (Name, Value) pairs can be provided as a vector.

Constructors**LDIFSubstitute()****Syntax:**

```
public LDIFSubstitute()
```

Methods**Vector substitute****Syntax:**

```
public static java.util.Vector substitute(Vector ldifEntry, String srchStr,  
String repStr)
```

Description:

Search and replace the substitution variables in an LDIF entry contained in a vector.

Parameters:

ldifEntry - the LDIF entry with elements as the attributes

srchStr - The substitution variable name

repStr - The substitution variable value

Returns:

Vector LDIF Entry after applying the substitution.

Vector substitute

Syntax:

```
public static java.util.Vector substitute(Vector ldifEntry, Vector sAndRep)
```

Description:

Search and replace the substitution variables in an LDIF entry contained in a vector.

Parameters:

ldifEntry - the LDIF entry with elements as the attributes

sAndRep - the vector containing substitution variables name-value pairs

Returns:

Vector LDIF Entry after applying the substitution.

oracle.ldap.util.LDIFWriter

Syntax:

```
public class oracle.ldap.util.LDIFWriter
```

Description:

LDAP Data Interchange Format (LDIF) is a file format used to represent the directory entries. The data from the directory can be exported in this format and can be imported into another directory server. The import and export of the directory data from an LDAP server can describe a set of changes that can be applied to data in a directory. This format is described in the Internet draft, The LDAP Data Interchange Format (LDIF) - RFC-2849.

This class provides a set of methods to write an LDIF entry to the file.

Constructors

LDIFWriter()

Syntax:

```
public LDIFWriter()
```

Description:

Default Constructor. Creates a writer stream to the standard output. Since the file name is not specified, the LDIF data is redirected to the standard output. The output data is written in UTF8 format. If the attribute value contains more characters than max line len, then it will not be wrapped.

Throws:

IOException - An I/O error has occurred

LDIFWriter(String file)**Syntax:**

```
public LDIFWriter(String file)
```

Description:

Creates a writer stream to the specified file for writing the LDIF data. If the attribute value contains more characters than max line length, then it will not be wrapped.

Parameters:

file - The name of the LDIF file

Throws:

IOException - An I/O error has occurred

LDIFWriter(File fileObj)**Syntax:**

```
public LDIFWriter(File fileObj)
```

Description:

Creates a writer stream to the specified file for writing the LDIF data. If the attribute value contains more characters than max line length, then it will not be wrapped.

Parameters:

fileObj - The file object of the LDIF file

Throws:

IOException - An I/O error has occurred

LDIFWriter(OutputStream out)**Syntax:**

```
public LDIFWriter(OutputStream out)
```

Description:

Creates a writer stream using the specified output stream object for writing the LDIF data. If the attribute value contains more characters than max line length, then it will not be wrapped.

Parameters:

out - Stream onto which the LDIF data needs to be written

Throws:

IOException - An I/O error has occurred

LDIFWriter(String file, boolean wrap)**Syntax:**

```
public LDIFWriter(String file, boolean wrap)
```

Description:

Creates a writer stream to the specified file for writing the LDIF data. If the attribute value contains more characters than max line length, then it will be wrapped to next line with a space as the first character.

Parameters:

file - The name of the LDIF file

wrap - If true and the attribute value has more characters than max line length, then the line will be wrapped

Throws:

IOException - An I/O error has occurred

LDIFWriter(File fileObj, boolean wrap)**Syntax:**

```
public LDIFWriter(File fileObj, boolean wrap)
```

Description:

Creates a writer stream to the specified file for writing the LDIF data. If the attribute value contains more characters than max line length, then it will be wrapped to next line with a space as the first character.

Parameters:

fileObj - The file object of the LDIF file

wrap - If true and the attribute value has more characters than max line length, then the line will be wrapped

Throws:

IOException - An I/O error has occurred

LDIFWriter(OutputStream out, boolean wrap)

Syntax:

```
public LDIFWriter(OutputStream out, boolean wrap)
```

Description:

Creates a writer stream using the specified output stream object for writing the LDIF data. If the attribute value contains more characters than max line length, then it will be wrapped to next line with a space as the first character.

Parameters:

out - stream on to which the LDIF data needs to be written

wrap - if true and the attribute value has more characters than max line length, then the line will be wrapped

Throws:

IOException - An I/O error has occurred

Methods

setMaxLineLen(int maxLineLen)

Syntax:

```
public void setMaxLineLen(int maxLineLen)
```

Description:

Use this method to set the maximum number of characters that can be written in a line.

Parameters:

maxLineLen - The maximum number of characters in a line

setWrap(boolean wrap)

Syntax:

```
public void setWrap(boolean wrap)
```

Description:

Use this method to specify if the wrapping of the attribute value should be done or not.

Parameters:

wrap - if true and the attribute value has more characters than max line length, then the line will be wrapped

writeEntry(Vector vEntry)

Syntax:

```
public void writeEntry(Vector vEntry)
```

Description:

Use this method to write an LDIF entry to the file.

Parameters:

vEntry - A vector containing the Attribute Names:Values as elements

Throws:

IOException - If an I/O error occurs

writeComment(String comment)

Syntax:

```
public void writeComment(String comment)
```

Description:

Use this method to add a comment line to the LDIF file.

Parameters:

comment - The comment string which is to be added to the LDIF file

Throws:

IOException - If an I/O error occurs

synchronized void close()

Syntax:

```
public synchronized void close()
```

Description:

Closes the stream.

Throws:

IOException - If an error occurs

oracle.ldap.util.Property

Syntax:

```
public class oracle.ldap.util.Property
```

Description:

This class represents a particular property in a PropertySet. In other words, it represents a particular attribute of an entry's attribute set.

Methods

int size()

Syntax:

```
public final int size()
```

Description:

Returns the size of this property, the number of values of the returned attribute.

Returns:

An int indicating the number of values belonging to this property. It can return 0.

getName()

Syntax:

```
public final java.lang.String getName()
```

Description:

Returns the name of this property, the name of the attribute this property represents.

Returns:

A string representing the name of this property.

Object getValue(int i)

Syntax:

```
public final java.lang.Object getValue(int i)
```

Description:

Returns the i-th value of this property - the i-th attribute value. An object is returned. The user must type-cast this appropriately.

Parameters:

int - The index of the value to be retrieved.

Returns:

The i-th value of this property.

oracle.ldap.util.PropertySet

Syntax:

```
public class oracle.ldap.util.PropertySet
```

Description:

This class represents a particular PropertySet in a PropertySetCollection. In other words, it represents a particular search result entry from a collection of search results.

Methods

isEmpty()

Syntax:

```
public final boolean isEmpty()
```

Description:

Returns true if the property set does not contain any properties, false otherwise.

Returns:

Boolean indicating whether the property set is empty or not.

size()

Syntax:

```
public final int size()
```

Description:

Returns the size of this property set, the number of returned attributes for this particular search result entry.

Returns:

An int indicating the number of properties belonging to this property set. It can return 0.

getAttributeNames()

Syntax:

```
public final java.lang.String[] getAttributeNames()
```

Description:

Returns an array of the string containing the name of all the properties. All the attribute names are returned with this particular search entry.

Returns:

A string array containing all the property names.

getProperty(int i)

Syntax:

```
public final oracle.ldap.util.Property getProperty(int i)
```

Description:

Returns the i-th property of this property set, the i-th attribute of this search entry.

Parameters:

i - the index of the property to be retrieved.

Returns:

A property representing the i-th property.

getProperty(String attrID)

Syntax:

```
public final oracle.ldap.util.Property getProperty(String attrID)
```

Description:

Returns the property identified by the attrID. The attrID is the attribute name.

Parameters:

attrID - The attribute name to be retrieved.

Returns:

A property with attribute name attrID.

getDN()

Syntax:

```
public final java.lang.String getDN()
```

Description:

Returns the name of this property set, the DN of the search entry represented by this property set.

Returns:

A string representing the DN of the property set.

oracle.ldap.util.PropertySetCollection

Syntax:

```
public class oracle.ldap.util.PropertySetCollection
```

Description:

This class represents a collection of PropertySets. In other words, it represents a set of search result entries from a given search.

Methods**isEmpty()****Syntax:**

```
public final boolean isEmpty()
```

Description:

Returns `true` if the property set collection does not contain any property set, `false` otherwise.

Returns:

Boolean indicating whether the property set is empty or not.

size()**Syntax:**

```
public final int size()
```

Description:

Returns the size of the property set collection, that is, the number of search entries in the search result.

Returns:

An `int` indicating the number of property sets in the collection.

getDns()**Syntax:**

```
public final java.lang.String[] getDns()
```

Description:

Returns a string array containing the name of all the property sets, that is, the DN's of all the search entries in this search result.

Returns:

A string array containing all the property set names.

getPropertySet(int i)**Syntax:**

```
public final oracle.ldap.util.PropertySet getPropertySet(int i)
```

Description:

Returns the i-th property set of this property set collection, that is, the i-th search entry of this search result.

Parameters:

int - i - The index of the property set to be retrieved.

Returns:

A PropertySet representing the i-th property set.

getPropertySet(String dn)**Syntax:**

```
public final oracle.ldap.util.PropertySet getPropertySet(String dn)
```

Description:

Returns the property set identified by dn, that is, the search entry with the given DN.

Parameters:

String - dn - The dn of the property set to be retrieved.

Returns:

A PropertySet with the given dn.

oracle.ldap.util.Subscriber

Syntax:

```
public class oracle.ldap.util.Subscriber
```

Constructors

Subscriber(DirContext ctx, int inSubIdType, String inSubIdName, boolean validate)

Syntax:

```
Subscriber(DirContext ctx, int inSubIdType, String inSubIdName, boolean  
validate)
```

Description:

Constructs a subscriber.

Parameters:

ctx - a valid DirContext

inSubIdType - The type of subscriber ID being used. Use one of Util.IDTYPE_DN, Util.IDTYPE_SIMPLE, or Util.IDTYPE_GUID.

inSubIdName - Subscriber ID. If this value is null and inSubIdType is Util.IDTYPE_DN, then default subscriber will be used. Otherwise, a null will cause an exception.

validate - Set to true to validate the user by the constructor.

Methods

getProperties(DirContext ctx, String[] attrList)

Syntax:

```
public oracle.ldap.util.PropertySetCollection getProperties(DirContext ctx,  
String[] attrList)
```

Description:

Retrieves selected attributes associated with this subscriber.

Parameters:

ctx - A valid DirContext

attrList - An array of attributes to be retrieved

getExtendedProperties**Syntax:**

```
public oracle.ldap.util.PropertySetCollection getExtendedProperties(DirContext
ctx, int propType, String[] attrList, String filter)
```

Description:

Retrieves extended properties under the oracle context of this subscriber. Currently, only "Common" properties and properties are supported.

Parameters:

ctx - A valid DirContext

propType - Use one of EXTPROPTYPE_COMMON, EXTPROPTYPE_RESOURCE_ACCESS_TYPE, or EXTPROPTYPE_DEFAULT_RAD.

attrList - An array of attributes to be retrieved.

filter - A search filter to narrow search criteria. For example, use (orclResourceTypeName=OracleDB) with EXTPROPTYPE_RESOURCE_ACCESS_TYPE to get only those properties of OracleDB.

resolve(DirContext ctx)**Syntax:**

```
public void resolve(DirContext ctx)
```

Description:

Validates the subscriber by identifying its DN.

Parameters:

ctx - A valid DirContext

getDN(DirContext ctx)**Syntax:**

```
public java.lang.String getDN(DirContext ctx)
```

Description:

Returns the DN of the subscriber (resolves the name if necessary).

Parameters:

ctx - A valid DirContext

getDn(DirContext ctx)

Syntax:

```
public java.lang.String getDn(DirContext ctx)
```

Description:

Returns the DN of this subscriber.

Fields**EXTPROPTYPE_COMMON****Syntax:**

```
public static EXTPROPTYPE_COMMON
```

Description:

Extended Preference Type to be used with `getExtendedProperties()` - Common Subscriber properties.

EXTPROPTYPE_RESOURCE_ACCESS_TYPE**Syntax:**

```
public static EXTPROPTYPE_RESOURCE_ACCESS_TYPE
```

Description:

Extended Preference Type to be used with `getExtendedProperties()` - Resource Access Type.

EXTPROPTYPE_DEFAULT_RAD**Syntax:**

```
public static EXTPROPTYPE_DEFAULT_RAD
```

Description:

Extended Preference Type to be used with `getExtendedProperties()` - Default User Extended Properties.

oracle.ldap.util.User

Syntax:

```
public class oracle.ldap.util.User
```

Constructors

User(DirContext ctx, int inUserIdType, String inUserName, subscriber inSubscriber, boolean validate)

Syntax:

```
public User(DirContext ctx, int inUserIdType, String inUserName, int  
inSubIdType, String inSubIdName, boolean validate)
```

Description:

Constructs a user.

Parameters:

ctx - A valid DirContext

inUserIdType - The type of user ID being used. Use one of Util.IDTYPE_DN, Util.IDTYPE_SIMPLE, or Util.IDTYPE_GUID.

inUserName - User ID

inSubscriber - A valid subscriber object

validate - Set to true to validate the user by the constructor

User(DirContext ctx, int inUserIdType, String inUserName, int inSubIdType, String inSubIdName, boolean validate)

Syntax:

```
public User(DirContext ctx, int inUserIdType, String inUserName, int  
inSubIdType, String inSubIdName, boolean validate)
```

Description:

Constructs a user.

Parameters:

ctx - A valid DirContext

`inUserIdType` - The type of user ID being used. Use one of `Util.IDTYPE_DN`, `Util.IDTYPE_SIMPLE`, or `Util.IDTYPE_GUID`.

`inUserName` - User ID

`inSubIdType` - The type of subscriber ID being used. Use one of `Util.IDTYPE_DN`, `Util.IDTYPE_SIMPLE`, or `Util.IDTYPE_GUID`.

`inSubIdName` - Subscriber ID

`validate` - Set to true to validate the user by the constructor

Methods

`getExtendedProperties`

Syntax:

```
public oracle.ldap.util.PropertySetCollection getExtendedProperties(DirContext  
ctx, int propType, String[] attrList, String filter)
```

Description:

Returns a property set collection of the extended properties of this user based on the property type specified.

Parameters:

`ctx` - A valid `DirContext`

`propType` - Currently only supporting `EXTPROPTYPE_RESOURCE_ACCESS_DESCRIPTOR`

`attrList` - A string array of attributes to be retrieved. Null indicates that all attributes should be retrieved. An empty array indicates that none should be retrieved.

`filter` - A search filter to identify any particular applications whose properties are to be retrieved.

Returns:

A `PropertySetCollection` containing the results.

`getExtendedProperties`

Syntax:

```
public oracle.ldap.util.PropertySetCollection getExtendedProperties(DirContext  
ctx,int propType, String[] attrList)
```

Description:

Returns a property set collection of all the extended properties of this user based on the property type specified.

Parameters:

ctx - A valid DirContext

proptype - Currently only supporting EXTPROPTYPE_RESOURCE_ACCESS_DESCRIPTOR.

attrList - A string array of attributes to be retrieved. Null indicates that all attributes should be retrieved. An empty array indicates that none should be retrieved.

Returns:

A PropertySetCollection containing the results.

getProperties**Syntax:**

```
public oracle.ldap.util.PropertySetCollection getProperties(DirContext ctx,  
String[] attrList)
```

Description:

Retrieves selected attributes associated with this user.

Parameters:

ctx - A valid DirContext

attrList - An array of attributes to be retrieved. Null indicates that all attributes should be retrieved. An empty array indicates that none should be retrieved.

setProperties**Syntax:**

```
public void setProperties(DirContext ctx, ModificationItem[] mods)
```

resolve(DirContext ctx)**Syntax:**

```
public void resolve(DirContext ctx)
```

Description:

Validates the user by identifying its DN.

Parameters:

ctx - A valid DirContext

getDn(DirContext ctx)**Syntax:**

```
public java.lang.String getDn(DirContext ctx)
```

Description:

Returns the DN of this user.

locateSubscriber(DirContext ctx)**Syntax:**

```
public java.lang.String locateSubscriber(DirContext ctx)
```

Description:

Locates the subscriber this user belongs to.

authenticateUser**Syntax:**

```
public void authenticateUser(DirContext ctx, int authType, Object cred)
```

Description:

Authenticates the user using the appropriate credentials.

Parameters:

ctx - A valid DirContext

authType - User.CREDTYPE_PASSWD is currently supported

cred - The credentials based on the authType

Fields

CREDTYPE_PASSWD

Syntax:

```
public static CREDTYPE_PASSWD
```

Description:

Uses user password to authenticate.

EXTPROPTYPE_RESOURCE_ACCESS_DESCRIPTOR

Syntax:

```
public static EXTPROPTYPE_RESOURCE_ACCESS_DESCRIPTOR
```

Description:

Extended Preference Type to be used with `getExtendedProperties()` - Resource Access Descriptor.

oracle.ldap.util.Util

Syntax:

```
public class oracle.ldap.util.Util
```

Constructors

Util()

Syntax:

```
public Util()
```

Methods

PropertySetCollection getEntryDetails

Syntax:

```
public static oracle.ldap.util.PropertySetCollection getEntryDetails(DirContext ctx, String base, String filter, int scope, String[] attrList)
```

setEntryDetails**Syntax:**

```
public static void setEntryDetails(DirContext ctx, String base,
ModificationItem[] mods)
```

authenticateUser**Syntax:**

```
public static void authenticateUser(DirContext ctx, User curUser, int authType,
Object cred)
```

Description:

Authenticates the user using the appropriate credentials.

Parameters:

ctx - A valid DirContext

authType - Util.CREDTYPE_PASSWD is currently supported

cred - The credentials based on the authType

getSubscriberDn**Syntax:**

```
public static java.lang.String getSubscriberDn(DirContext ctx, String subId, int
subIdType)
```

Description:

Returns the DN of the given subscriber. If DN is used as the subscriber ID, then simply look up the DN to validate it.

Parameters:

ctx - A valid DirContext

subId - Subscriber ID

subIdType - The type of subscriber ID being used. Use one of IDTYPE_DN, IDTYPE_SIMPLE, or Util.IDTYPE_GUID.

getUserDn**Syntax:**

```
public static java.lang.String getUserDn(DirContext ctx, String userId, int
userIdType, String subscriberDN)
```

Description:

Returns the DN of the given user. If DN is used as the user ID, then simply look up the DN to validate it. If subscriber DN is null, the default subscriber will be used.

Parameters:

ctx - A valid DirContext

userId - User ID

userIdType - The type of user ID being used. Use one of IDTYPE_DN, IDTYPE_SIMPLE, or Util.IDTYPE_GUID.

oracle.ldap.util.PropertySetCollection getGroupMembership**Syntax:**

```
public static oracle.ldap.util.PropertySetCollection
getGroupMembership(DirContext ctx, User curUser, String[] attrList, boolean
nested)
```

Description:

Returns a list of groups the user belongs to directly or indirectly.

Parameters:

ctx - A valid DirContext curUser - A valid user object

curUser - A valid user object

attrList - A string array of attributes to be retrieved from each group

nested - Set to true to look for nested membership. Otherwise, only direct memberships are returned.

vector2StrArray(Vector list)**Syntax:**

```
public static java.lang.String[] vector2StrArray(vector list)
```

normalizeDN(String inDn)**Syntax:**

```
public static java.lang.String normalizeDN(String inDn)
```

getDASUrl(DirContext ctx, String urlTypeDN)**Syntax:**

```
public static java.lang.String getDASUrl(DirContext ctx, String urlTypeDN)
```

Description:

Returns a particular DAS URL identified by the urlTypeDN.

Parameters:

ctx - A valid DirContext

urlTypeDN - Use one of Util.DASURL_* which represents a particular URL type.

Returns:

A string representing the URL.

java.util.Hashtable getAllDASUrl(DirContext ctx)**Syntax:**

```
public static java.util.Hashtable getAllDASUrl(DirContext ctx)
```

Description:

Returns a hashtable containing all the DAS URLs. Each individual URL can then be retrieved from the hashtable using Util.DASURL_* as the key to identify the URL type.

Parameters:

ctx - A valid DirContext

Returns:

A hashtable containing all the DAS URLs

Methods

printResults(PropertySetCollection resultSet)

Syntax:

```
public static void printResults(PropertySetCollection resultSet)
```

Description:

Prints the entries represented by the PropertySetCollection in LDIF format.

Parameters:

resultSet - A valid PropertySetCollection

getDefaultSubscriber

Syntax:

```
public static java.lang.String[] getDefaultSubscriber()
```

createDefaultSubscriber

Syntax:

```
public static void createDefaultSubscriber(DirContext ctx, String ohome, String domain, String subscriber)
```

subAndLoadLdif

Syntax:

```
public static void subAndLoadLdif(DirContext ctx, String filename, Vector subVector)
```

checkInterfaceVersion(String intVersion)

Syntax:

```
public static boolean checkInterfaceVersion(String intVersion)
```

Description:

This method checks if the interface version given is supported with the version of the current API.

Parameters:

String - Interface version

Fields

API_VERSION

Syntax:

```
public static API_VERSION
```

Description:

The API version number.

INTERFACE_VERSION

Syntax:

```
public static INTERFACE_VERSION
```

Description:

The interface version number.

IDTYPE_DN

Syntax:

```
public static IDTYPE_DN
```

Description:

The ID is used as a DN.

IDTYPE_SIMPLE

Syntax:

```
public static IDTYPE_SIMPLE
```

Description:

The ID used is a simple ID.

IDTYPE_GUID

Syntax:

```
public static IDTYPE_GUID
```

Description:

The ID used is a GUID.

IDTYPE_DEFAULT

Syntax:

```
public static IDTYPE_DEFAULT
```

Description:

Use default value.

PROPERTIES_ENTRY

Syntax:

```
public static PROPERTIES_ENTRY
```

Description:

User entry properties.

PROPERTIES_DETACHED

Syntax:

```
public static PROPERTIES_DETACHED
```

Description:

User detached properties.

CREDTYPE_PASSWD

Syntax:

```
public static CREDTYPE_PASSWD
```

Description:

Using user password to authenticate.

DASURL_BASE

Syntax:

```
public static DASURL_BASE
```

Description:

DAS URL type - Base URL

DASURL_CREATE_USER**Syntax:**

```
public static DASURL_CREATE_USER
```

Description:

DAS URL type - Create User

DASURL_EDIT_GROUP**Syntax:**

```
public static DASURL_EDIT_GROUP
```

Description:

DAS URL type - Edit Group

DASURL_EDIT_GROUP_GIVEN_GUID**Syntax:**

```
public static DASURL_EDIT_GROUP_GIVEN_GUID
```

Description:

DAS URL type -Edit Group Given GUID

DASURL_GROUP_SEARCH**Syntax:**

```
public static DASURL_GROUP_SEARCH
```

Description:

DAS URL type - Group Search

DASURL_EDIT_USER**Syntax:**

```
public static DASURL_EDIT_USER
```

Description:

DAS URL type - Edit User

DASURL_GROUP_LOV

Syntax:

```
public static DDASURL_GROUP_LOV
```

Description:

DAS URL type -Group LOV

DASURL_DELETE_USER

Syntax:

```
public static DASURL_DELETE_USER
```

Description:

DAS URL type - Delete User

DASURL_USER_PRIVILEGE

Syntax:

```
public static DASURL_USER_PRIVILEGE
```

Description:

DAS URL type - User Privilege

DASURL_CREATE_GROUP

Syntax:

```
public static DASURL_CREATE_GROUP
```

Description:

DAS URL type - Create Group

DASURL_USER_SEARCH

Syntax:

```
public static DASURL_USER_SEARCH
```

Description:

DAS URL type - User Search

DASURL_ACCOUNT_INFO**Syntax:**

```
public static DASURL_ACCOUNT_INFO
```

Description:

DAS URL type - Account Info

DASURL_EDIT_USER_GIVEN_GUID**Syntax:**

```
public static DASURL_EDIT_USER_GIVEN_GUID
```

Description:

DAS URL type - Edit User Given GUID

DASURL_DELETE_USER_GIVEN_GUID**Syntax:**

```
public static DASURL_DELETE_USER_GIVEN_GUID
```

Description:

DAS URL type - Delete User Given GUID

DASURL_DELETE_GROUP_GIVEN_GUID**Syntax:**

```
public static DASURL_DELETE_GROUP_GIVEN_GUID
```

Description:

DAS URL type - Delete Group Given GUID

DASURL_GROUP_PRIVILEGE

Syntax:

```
public static DASURL_GROUP_PRIVILEGE
```

Description:

DAS URL type - Group Privilege

DASURL_USER_PRIVILEGE_GIVEN_GUID

Syntax:

```
public static DASURL_USER_PRIVILEGE_GIVEN_GUID
```

Description:

DAS URL type - User Privilege Given GUID

DASURL_PASSWORD_CHANGE

Syntax:

```
public static DASURL_PASSWORD_CHANGE
```

Description:

DAS URL type - Password Change

DASURL_USER_LOV

Syntax:

```
public static DASURL_USER_LOV
```

Description:

DAS URL type - Password Change

DASURL_GROUP_PRIVILEGE_GIVEN_GUID

Syntax:

```
public static DASURL_GROUP_PRIVILEGE_GIVEN_GUID
```

Description:

DAS URL type - Group Privilege Given GUID

DASURL_DELETE_GROUP**Syntax:**

```
public static DASURL_DELETE_GROUP
```

Description:

DAS URL type - Delete Group

DASURL_CREATE_RESOURCE**Syntax:**

```
public static DASURL_CREATE_RESOURCE
```

Description:

DAS URL type - Create Resource

oracle.ldap.util.jndi.ConnectionUtil**Syntax:**

```
public class oracle.ldap.util.jndi.ConnectionUtil
```

Constructors**ConnectionUtil()****Syntax:**

```
public ConnectionUtil()
```

Methods**discoverSSLPort****Syntax:**

```
public static java.lang.String discoverSSLPort(String host, String port, String bindDn, String bindPwd)
```

Description:

This method will connect to an existing non-SSL OID using the connect information provided and obtain the SSL connect information in the default configset 1. The SSL OID server need not be running at this point.

Parameters:

host - The host name where the non-SSL OID is running

port - The port number on which the non-SSL OID is running

bindDN - The bind DN (for example, cn=orcladmin)

bindPwd - The bind password

getDefaultDirCtx**Syntax:**

```
public static javax.naming.directory.InitialDirContext getDefaultDirCtx(String host, String port, String bindDN, String bindPwd)
```

Description:

Returns an InitialDirContext using the connect information provided. The corresponding non-SSL OID server must be running. For SSL connection, please use getSSLDIRCtx instead.

Parameters:

host - The host name where the non-SSL OID is running

port - The port number on which the non-SSL OID is running

bindDN - The bind DN (for example, cn=orcladmin)

bindPwd - the bind password

getSSLDIRCtx**Syntax:**

```
public static javax.naming.directory.InitialDirContext getSSLDIRCtx(String host, String port, String bindDN, String bindPwd)
```

Description:

Returns an InitialDirContext using the connect information provided. Use this only if this is an SSL connection. The corresponding SSL OID server must be running.

Parameters:

host - The host name where the non-SSL OID is running

port - The port number on which the non-SSL OID is running

bindDN - The bind DN (for example, cn=orcladmin)

bindPwd - the bind password

Exceptions

This section explains exceptions. It contains these topics:

- [oracle.ldap.util.AcctTotallyLockedException](#)
- [oracle.ldap.util.AuthFailureException](#)
- [oracle.ldap.util.AuthPasswdChangeWarningException](#)
- [oracle.ldap.util.AuthPasswdExpiredException](#)
- [oracle.ldap.util.GeneralErrorException](#)
- [oracle.ldap.util.InvalidLDIFRecordException](#)
- [oracle.ldap.util.InvalidParameterException](#)
- [oracle.ldap.util.InvalidRootOrclctxException](#)
- [oracle.ldap.util.InvalidSubscriberOrclctxException](#)
- [oracle.ldap.util.LoginPolicyFailureException](#)
- [oracle.ldap.util.MigrationException](#)
- [oracle.ldap.util.MultipleSubscriberException](#)
- [oracle.ldap.util.MultipleUserException](#)
- [oracle.ldap.util.NoGroupMembersException](#)
- [oracle.ldap.util.NoRootOrclctxException](#)
- [oracle.ldap.util.NoSubscriberOrclctxException](#)
- [oracle.ldap.util.NoSuchGroupException](#)
- [oracle.ldap.util.NoSuchSubscriberException](#)
- [oracle.ldap.util.NoSuchUserException](#)
- [oracle.ldap.util.ParameterException](#)

- [oracle.ldap.util.PasswdExpiredException](#)
- [oracle.ldap.util.SetPropertiesException](#)
- [oracle.ldap.util.SubscriberNotFoundException](#)
- [oracle.ldap.util.UtilException](#)

oracle.ldap.util.AcctIPLockedException

Syntax:

```
public class oracle.ldap.util.AcctIPLockedException extends
oracle.ldap.util.UtilException
```

Constructors

AcctIPLockedException()

Syntax:

```
public AcctIPLockedException()
```

AcctIPLockedException(String s)

Syntax:

```
public AcctIPLockedException(String s)
```

oracle.ldap.util.AcctTotallyLockedException

Syntax:

```
public class oracle.ldap.util.AcctTotallyLockedException extends
oracle.ldap.util.UtilException
```

Constructors

AcctTotallyLockedException()

Syntax:

```
public AcctTotallyLockedException()
```

AcctTotallyLockedException(String s)**Syntax:**

```
public AcctTotallyLockedException(String s)
```

oracle.ldap.util.AuthFailureException**Syntax:**

```
public class oracle.ldap.util.AuthFailureException extends  
oracle.ldap.util.UtilException
```

oracle.ldap.util.AuthPasswdChangeWarningException**Syntax:**

```
public class oracle.ldap.util.AuthPasswdChangeWarningException extends  
oracle.ldap.util.UtilException
```

Constructors**AuthPasswdChangeWarningException()****Syntax:**

```
public AuthPasswdChangeWarningException()
```

AuthPasswdChangeWarningException(String s)**Syntax:**

```
public AuthPasswdChangeWarningException(String s)
```

oracle.ldap.util.AuthPasswdExpiredException

```
public class oracle.ldap.util.AuthPasswdExpiredException extends  
oracle.ldap.util.UtilException
```

Constructors

AuthPasswdExpiredException()

Syntax:

```
AuthPasswdExpiredException()
```

AuthPasswdExpiredException(String s)

Syntax:

```
AuthPasswdExpiredException(String s)
```

oracle.ldap.util.GeneralErrorException

Syntax:

```
public class oracle.ldap.util.GeneralErrorException extends  
oracle.ldap.util.UtilException
```

Description:

This exception is thrown when a general error is encountered.

Constructors

GeneralErrorException()

Syntax:

```
public GeneralErrorException()
```

Description:

Constructs a `GeneralErrorException` with no detail message.

GeneralErrorException(String s)

Syntax:

```
public GeneralErrorException(String s)
```

Description:

Constructs a `GeneralErrorException` with the specified detail message.

Parameters:

s - the detail message

oracle.ldap.util.InvalidLDIFRecordException**Syntax:**

```
public class oracle.ldap.util.InvalidLDIFRecordException extends
    java.lang.RuntimeException
```

Description:

An object of this class will be thrown when an error occurs during LDIF record parsing.

Constructors**InvalidLDIFRecordException()****Syntax:**

```
public InvalidLDIFRecordException()
```

Description:

Constructs an `InvalidLDIFRecordException` with no detail message.

InvalidLDIFRecordException(int lineNo, String s)**Syntax:**

```
public InvalidLDIFRecordException(int lineNo, String s)
```

Description:

Constructs an `InvalidLDIFRecordException` with the specified detail message.

Parameters:

s - the detail message

Methods**printStackTrace()**

Syntax:

```
void printStackTrace()
```

printStackTrace(PrintStream pout)

Syntax:

```
void printStackTrace(PrintStream pout)
```

printStackTrace(PrintWriter wout)

Syntax:

```
void printStackTrace(PrintWriter wout)
```

oracle.ldap.util.InvalidParameterException

Syntax:

```
public class oracle.ldap.util.InvalidParameterException extends  
java.lang.Exception
```

Description:

An object of this class will be thrown when an error occurs during input parameter parsing.

Constructors

InvalidParameterException()

Syntax:

```
public InvalidParameterException()
```

Description:

Constructs an `InvalidParameterException` with no detail message.

InvalidParameterException(String s)

Syntax:

```
public InvalidParameterException(String s)
```

Description:

Constructs an `InvalidParameterException` with the specified detail message.

Parameters

s - the detail message

oracle.ldap.util.InvalidRootOrclctxException**Syntax:**

```
public class oracle.ldap.util.InvalidRootOrclctxException extends  
oracle.ldap.util.UtilException
```

Description:

This exception will be thrown when an invalid root oracle context is encountered while searching within the root oracle context.

Constructors**InvalidRootOrclctxException()****Syntax:**

```
public InvalidRootOrclctxException()
```

Description:

Constructs an `InvalidRootOrclctxException` with no detail message.

InvalidRootOrclctxException(String s)**Syntax:**

```
public InvalidRootOrclctxException(String s)
```

Description:

Constructs an `InvalidRootOrclctxException` with the specified detail message.

Parameters:

s - the detail message

oracle.ldap.util.InvalidSubscriberOrclctxException

Syntax:

```
public class oracle.ldap.util.InvalidSubscriberOrclctxException extends  
oracle.ldap.util.UtilException
```

Description:

This exception will be thrown when an invalid oracle context within the subscriber is encountered.

Constructors

InvalidSubscriberOrclctxException()

Syntax:

```
public InvalidSubscriberOrclctxException()
```

Description:

Constructs an `InvalidSubscriberOrclctxException` with no detail message.

InvalidSubscriberOrclctxException(String s)

Syntax:

```
public InvalidSubscriberOrclctxException(String s)
```

Description:

Constructs an `InvalidSubscriberOrclctxException` with the specified detail message.

Parameters:

s - the detail message

oracle.ldap.util.LoginPolicyFailureException

Syntax:

```
public class oracle.ldap.util.LoginPolicyFailureException extends  
oracle.ldap.util.UtilException
```


Constructors

LoginPolicyFailureException()

Syntax:

```
public LoginPolicyFailureException()
```

LoginPolicyFailureException(String s)

Syntax:

```
public LoginPolicyFailureException(String s)
```

oracle.ldap.util.MigrationException

Syntax:

```
public class oracle.ldap.util.MigrationException extends java.lang.Exception
```

Description:

An object of this class will be thrown when a migration exception occurs.

Constructors

MigrationException()

Syntax:

```
public MigrationException()
```

Description:

Constructs a `MigrationException` with no detail message.

MigrationException(String s)

Syntax:

```
public MigrationException(String s)
```

Description:

Constructs a `MigrationException` with the specified detail message.

Parameters:

s - the detail message

oracle.ldap.util.MultipleSubscriberException

Syntax:

```
public class oracle.ldap.util.MultipleSubscriberException extends  
oracle.ldap.util.UtilException
```

Description:

This exception is thrown when multiple subscribers of the same ID are encountered under the subscriber search base.

Constructors

MultipleSubscriberException()

Syntax:

```
public MultipleSubscriberException()
```

Description:

Constructs a `MultipleSubscriberException` with no detail message.

MultipleSubscriberException(String s)

Syntax:

```
public MultipleSubscriberException(String s)
```

Description:

Constructs a `MultipleSubscriberException` with the specified detail message.

Parameters:

s - the detail message

oracle.ldap.util.MultipleUserException

Syntax:

```
public class oracle.ldap.util.MultipleUserException extends
oracle.ldap.util.UtilException
```

Description:

This exception is thrown when multiple users of the same ID are encountered under the subscriber user search base.

Constructors**MultipleUserException()****Syntax:**

```
public MultipleUserException()
```

Description:

Constructs a `MultipleUserException` with no detail message.

MultipleUserException(String s)**Syntax:**

```
public MultipleUserException(String s)
```

Description:

Constructs a `MultipleUserException` with the specified detail message.

Parameters:

s - the detail message

oracle.ldap.util.NoGroupMembersException

Syntax:

```
public class oracle.ldap.util.NoGroupMembersException extends
oracle.ldap.util.UtilException
```

Description:

This exception is thrown if the user looks for their group membership but is not a unique member of any particular group.

Constructors

NoGroupMembersException()

Syntax:

```
public NoGroupMembersException()
```

Description:

Constructs a `NoGroupMemberException` with no detail message.

NoGroupMembersException(String s)

Syntax:

```
public NoGroupMembersException(String s)
```

Description:

Constructs a `NoGroupMemberException` with the specified detail message.

Parameters:

s - the detail message

oracle.ldap.util.NoRootOrclctxException

Syntax:

```
public class oracle.ldap.util.NoRootOrclctxException extends  
oracle.ldap.util.UtilException
```

Description:

This exception is thrown if the root oracle context does not exist.

Constructors

NoRootOrclctxException()

Syntax:

```
public NoRootOrclctxException()
```

Description:

Constructs a `NoRootOrclctxException` with no detail message.

NoRootOrclctxException(String s)

Syntax:

```
public NoRootOrclctxException(String s)
```

Description:

Constructs a `NoRootOrclctxException` with the specified detail message.

Parameters:

s- the detail message.

oracle.ldap.util.NoSubscriberOrclctxException

Syntax:

```
public class oracle.ldap.util.NoSubscriberOrclctxException extends  
oracle.ldap.util.UtilException
```

Description:

This exception is thrown when the search is unable to locate the oracle context within the subscriber.

Constructors

NoSubscriberOrclctxException()

Syntax:

```
public NoSubscriberOrclctxException()
```

Description:

Constructs a `NoSubscriberOrclctxException` with no detail message.

NoSubscriberOrclctxException(String s)

Syntax:

```
public NoSubscriberOrclctxException(String s)
```

Description:

Constructs a `NoSubscriberOrclctxException` with the specified detail message.

Parameters:

s - the detail message

oracle.ldap.util.NoSuchGroupException

Syntax:

```
public class oracle.ldap.util.NoSuchGroupException extends  
oracle.ldap.util.UtilException
```

Description:

This exception is thrown when the group object cannot be resolved because the group does not exist in the directory.

Constructors

NoSuchGroupException()

Syntax:

```
public NoSuchGroupException()
```

Description:

Constructs a `NoSuchGroupException` with no detail message.

NoSuchGroupException(String s)

Syntax:

```
public NoSuchGroupException(String s)
```

Description:

Constructs a `NoSuchGroupException` with the specified detail message.

Parameters:

s - the detail message

`oracle.ldap.util.NoSuchSubscriberException`

Syntax:

```
public class oracle.ldap.util.NoSuchSubscriberException extends  
oracle.ldap.util.UtilException
```

Description:

This exception is thrown when the subscriber object cannot be resolved because the subscriber does not exist under the subscriber search base.

Constructors**`NoSuchSubscriberException()`****Syntax:**

```
public NoSuchSubscriberException()
```

Description:

Constructs a `NoSuchSubscriberException` with no detail message.

`NoSuchSubscriberException(String s)`**Syntax:**

```
public NoSuchSubscriberException(String s)
```

Description:

Constructs a `NoSuchSubscriberException` with the specified detail message.

Parameters:

s - the detail message

oracle.ldap.util.NoSuchUserException

Syntax:

```
public class oracle.ldap.util.NoSuchUserException extends  
oracle.ldap.util.UtilException
```

Description:

This exception is thrown when the user object cannot be resolved because the user does not exist under the user search base of the specified subscriber.

Constructors

NoSuchUserException()

Syntax:

```
public NoSuchUserException()
```

Description:

Constructs a `NoSuchUserException` with no detail message.

NoSuchUserException(String s)

Syntax:

```
public NoSuchUserException(String s)
```

Description:

Constructs a `NoSuchUserException` with the specified detail message.

Parameters:

s - the detail message

oracle.ldap.util.ParameterException

Syntax:

```
public class oracle.ldap.util.ParameterException extends  
oracle.ldap.util.UtilException
```

Description:

This exception is thrown when an error occurs during input parameter parsing.

Constructors

ParameterException()

Syntax:

```
public ParameterException()
```

Description:

Constructs a `ParameterException` with no detail message.

ParameterException(String s)

Syntax:

```
public ParameterException(String s)
```

Description:

Constructs a `ParameterException` with the specified detail message.

Parameters:

s - the detail message

oracle.ldap.util.PasswdExpiredException

Syntax:

```
public class oracle.ldap.util.PasswdExpiredException extends  
oracle.ldap.util.UtilException
```

Constructors

PasswdExpiredException()

Syntax:

```
public PasswdExpiredException()
```

PasswdExpiredException(String s)

Syntax:

```
public PasswdExpiredException(String s)
```

oracle.ldap.util.SetPropertiesException

Syntax:

```
public class oracle.ldap.util.SetPropertiesException extends  
oracle.ldap.util.UtilException
```

Description:

This exception is thrown when modification cannot be carried out while using the `setProperties()` method.

Constructors

SetPropertiesException()

Syntax:

```
public SetPropertiesException()
```

Description:

Constructs a `SetPropertiesException` with no detail message.

SetPropertiesException(String s)

Syntax:

```
public SetPropertiesException(String s)
```

Description:

Constructs a `SetPropertiesException` with the specified detail message.

Parameters:

s - the detail message

oracle.ldap.util.SubscriberNotFoundException

Syntax:

```
public class oracle.ldap.util.SubscriberNotFoundException extends  
oracle.ldap.util.UtilException
```

Description:

This exception is thrown when the subscriber cannot be located under the subscriber search base.

Constructors**SubscriberNotFoundException()****Syntax:**

```
public SubscriberNotFoundException()
```

Description:

Constructs a `SubscriberNotFoundException` with no detail message.

SubscriberNotFoundException(String s)**Syntax:**

```
public SubscriberNotFoundException(String s)
```

Description:

Constructs a `SubscriberNotFoundException` with the specified detail message.

Parameters:

s - the detail message

oracle.ldap.util.UtilException**Syntax:**

```
public class oracle.ldap.util.UtilException extends java.lang.Exception
```

Constructors**UtilException()****Syntax:**

```
public UtilException()
```

UtilException(String s)

Syntax:

```
public UtilException(String s)
```

The DBMS_LDAP_UTL PL/SQL Package

This chapter introduces the DBMS_LDAP_UTL Package, which contains Oracle Extension utility functions. This chapter contains these topics:

- [Introduction](#)
- [DBMS_LDAP_UTL Reference](#)

Introduction

This section contains details about the DBMS_LDAP_UTL subprograms. The entries for each subprogram contain the following information:

Table 7–1 Function Entry Information

Term	Description
Syntax	A code snippet showing the syntax for calling the function, including the ordering and types of the parameters.
Description	A brief statement of the purpose of the function.
Comments	Detailed information about the function, if available. This can include restrictions on use of the function, or other information that may be useful when using the function in an application.
Parameters	A description of each of the function's parameters. This includes the parameter's mode. The mode of a parameter has the following possible values: <ul style="list-style-type: none">■ IN - A parameter that passes data to Oracle.■ OUT - A parameter that receives data from Oracle on this or a subsequent call.■ IN/OUT - A parameter that passes data on the call and receives data on the return from this or a subsequent call.
Return Values	Values returned by the function.
Usage Notes	Notes about using the specific function.
Related Functions	Related functions listed under the heading of See Also.

DBMS_LDAP_UTL Reference

This section contains information about the DBMS_LDAP_UTL functions. This section contains these topics:

- [Summary of Subprograms](#)
- [User-Related Subprograms](#)
- [Group-Related Subprograms](#)
- [Subscriber-Related Subprograms](#)
- [Property-Related Subprograms](#)
- [Miscellaneous Subprograms](#)

Summary of Subprograms

Table 7–2 DBMS_LDAP_UTL User-Related Subprograms

Function or Procedure	Purpose
Function <code>authenticate_user</code>	Authenticates a user against an LDAP server
Function <code>create_user_handle</code>	Creates a user handle
Function <code>set_user_handle_properties</code>	Associates the given properties to the user handle
Function <code>get_user_properties</code>	Retrieves user properties from an LDAP server
Function <code>set_user_properties</code>	Modifies the properties of a user
Function <code>get_user_extended_properties</code>	Retrieves user extended properties
Function <code>get_user_dn</code>	Retrieves a user DN
Function <code>check_group_membership</code>	Checks whether a user is member of the given group
Function <code>locate_subscriber_for_user</code>	Retrieves the subscriber for the given user
Function <code>get_group_membership</code>	Retrieves a list of groups of which the user is a member

Table 7–3 DBMS_LDAP_UTL Group-Related Subprograms

Function or Procedure	Purpose
Function <code>create_group_handle</code>	Creates a group handle
Function <code>set_group_handle_properties</code>	Associates the given properties with the group handle
Function <code>get_group_properties</code>	Retrieves group properties from an LDAP server
Function <code>get_group_dn</code>	Retrieves a group DN

Table 7-4 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 7-5 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.

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-15 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 7–6 *AUTHENTICATE_USER* Function Parameters

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

Return Values

Table 7-7 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.

Table 7-7 (Cont.) AUTHENTICATE_USER Function Return Values

Value	Description
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 7-8 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

Table 7–8 (Cont.) CREATE_USER_HANDLE Function Parameters

Parameter Name	Parameter Type	Parameter Description
user_id	VARCHAR2	The user ID representing the user entry.

Table 7–9**Return Values****Table 7–10 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 7–11 *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 7–12 *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 7–13 *GET_USER_PROPERTIES Function Parameters*

Parameter Name	Parameter Type	Parameter Description
<code>ld</code>	SESSION	A valid LDAP session handle.
<code>user_handle</code>	HANDLE	The user handle.
<code>attrs</code>	STRING_COLLECTION	The list of attributes to fetch for the user.
<code>ptype</code>	PLS_INTEGER	Type of properties to return. Valid values are as follows: - DBMS_LDAP_UTL.ENTRY_PROPERTIES, - DBMS_LDAP_UTL.NICKNAME_PROPERTY
<code>ret-pset_collection</code>	PROPERTY_SET_COLLECTION	The user details containing the attributes requested by the caller.

Return Values

Table 7–14 *GET_USER_PROPERTIES Function Return Values*

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.

Table 7–14 (Cont.) GET_USER_PROPERTIES Function Return Values

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.
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 7-15 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 7-16 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.

Table 7–16 (Cont.) SET_USER_PROPERTIES Function Return Values

Value	Description
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 7-17 *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_RESOURCE_ACCESS_DES
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 7-18 *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.

Table 7–18 (Cont.) GET_USER_EXTENDED_PROPERTIES Function Return Values

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 7–19 *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 7–20 *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 7–21 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 7–22 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.

Table 7–22 (Cont.) CHECK_GROUP_MEMBERSHIP Function Return Values

Value	Description
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 7–23 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 7–24 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.
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 7–25 *GET_GROUP_MEMBERSHIP Function Parameters*

Parameter Name	Parameter Type	Parameter Description
<code>ld</code>	SESSION	A valid LDAP session handle.
<code>user_handle</code>	HANDLE	The user handle.
<code>nested</code>	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
<code>attr_list</code>	STRING_COLLECTION	A list of attributes to be returned.
<code>ret_groups</code>	PROPERTY_SET_COLLECTION	A pointer to a pointer to an array of group entries.

Return Values

Table 7–26 *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-28 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 7–27 *CREATE_GROUP_HANDLE* Function Parameters

Parameter Name	Parameter Type	Parameter Description
<code>group_hd</code>	HANDLE	A pointer to a handle to a group.

Table 7–27 (Cont.) CREATE_GROUP_HANDLE Function Parameters

Parameter Name	Parameter Type	Parameter Description
group_type	PLS_INTEGER	The type of group ID that is passed. Valid values for this argument are as follows: - DBMS_LDAP_UTL.TYPE_DN, DBMS_LDAP_UTL.TYPE_GUID, DBMS_LDAP_UTL.TYPE_NICKNAME
group_id	VARCHAR2	The group ID representing the group entry.

Return Values**Table 7–28 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 7–29 *SET_GROUP_HANDLE_PROPERTIES* Function Parameters

Parameter Name	Parameter Type	Parameter Description
group_hd	HANDLE	A pointer to the handle to the group.
property_type	PLS_INTEGER	The type of property that is passed. Valid values for this argument are as follows: DBMS_LDAP_UTL.GROUP_HANDLE
property	HANDLE	The property describing the group entry.

Return Values

Table 7–30 *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 7–31 *GET_GROUP_PROPERTIES Function Parameters*

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

Return Values

Table 7–32 *GET_GROUP_PROPERTIES Function Return Values*

Value	Description
<code>DBMS_LDAP_UTL.SUCCESS</code>	On a successful completion.
<code>DBMS_LDAP_UTL.PARAM_ERROR</code>	Invalid input parameters.
<code>DBMS_LDAP_UTL.NO_SUCH_GROUP</code>	Group doesn't exist.

Table 7–32 GET_GROUP_PROPERTIES Function Return Values

Value	Description
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 7–33 *GET_GROUP_DN Function Parameters*

Parameter Name	Parameter Type	Parameter Description
<code>ld</code>	SESSION	A valid LDAP session handle.
<code>group_handle</code>	HANDLE	The group handle.
<code>dn</code>	VARCHAR2	The group DN.

Return Values

Table 7–34 *GET_GROUP_DN Function Return Values*

Value	Description
<code>DBMS_LDAP_UTL.SUCCESS</code>	On a successful completion.
<code>DBMS_LDAP_UTL.PARAM_ERROR</code>	Invalid input parameters.
<code>DBMS_LDAP_UTL.NO_SUCH_GROUP</code>	Group doesn't exist.
<code>DBMS_LDAP_UTL.MULTIPLE_GROUP_ENTRIES</code>	Multiple number of group DN entries exist in the directory for the given group.
<code>DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX</code>	Invalid Root Oracle Context.

Table 7–34 (Cont.) GET_GROUP_DN Function Return Values

Value	Description
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_util.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 its 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-15 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 7–35 *CREATE_SUBSCRIBER_HANDLE* Function Parameters

Parameter Name	Parameter Type	Parameter Description
<code>subscriber_hd</code>	HANDLE	A pointer to a handle to a subscriber.
<code>subscriber_type</code>	PLS_INTEGER	The type of subscriber ID that is passed. Valid values for this argument are: <code>DBMS_LDAP_UTL.TYPE_DN</code> , <code>DBMS_LDAP_UTL.TYPE_GUID</code> , <code>DBMS_LDAP_UTL.TYPE_NICKNAME</code> , <code>DBMS_LDAP_UTL.TYPE_DEFAULT</code>
<code>subscriber_id</code>	VARCHAR2	The subscriber ID representing the subscriber entry. This can be NULL if <code>subscriber_type</code> is as follows: <code>DBMS_LDAP_UTL.TYPE_DEFAULT</code> Then the default subscriber is fetched from Root Oracle Context.

Return Values

Table 7–36 *CREATE_SUBSCRIBER_HANDLE* Function Return Values

Value	Description
<code>DBMS_LDAP_UTL.SUCCESS</code>	On a successful completion.

Table 7–36 (Cont.) CREATE_SUBSCRIBER_HANDLE Function Return Values

Value	Description
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.GENERAL_ERROR	Other error.

See Also

DBMS_LDAP_UTL.get_subscriber_properties().

Function get_subscriber_properties

The function get_subscriber_properties() retrieves the subscriber properties for the given subscriber handle.

Syntax

```
FUNCTION get_subscriber_properties
(
  ld IN SESSION,
  subscriber_handle IN HANDLE,
  attrs IN STRING_COLLECTION,
  ptype IN PLS_INTEGER,
  ret_pset_coll OUT PROPERTY_SET_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters**Table 7–37 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.

Table 7–37 (Cont.) GET_SUBSCRIBER_PROPERTIES Function Parameters

Parameter Name	Parameter Type	Parameter Description
ret_pset_coll	PROPERTY_SET_ COLLECTON	The subscriber details containing the attributes requested by the caller.

Return Values**Table 7–38 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.
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 7–39 *GET_SUBSCRIBER_DN Function Parameters*

Parameter Name	Parameter Type	Parameter Description
<code>ld</code>	SESSION	A valid LDAP session handle.
<code>subscriber_handle</code>	HANDLE	The subscriber handle.
<code>dn</code>	VARCHAR2	The subscriber DN.

Return Values

Table 7–40 *GET_SUBSCRIBER_DN Function Return Values*

Value	Description
<code>DBMS_LDAP_UTL.SUCCESS</code>	On a successful completion.
<code>DBMS_LDAP_UTL.PARAM_ERROR</code>	Invalid input parameters.
<code>DBMS_LDAP_UTL.NO_SUCH_SUBSCRIBER</code>	Subscriber doesn't exist.
<code>DBMS_LDAP_UTL.MULTIPLE_SUBSCRIBER_ENTRIES</code>	Multiple number of subscriber DN entries exist in the directory for the given subscriber.
<code>DBMS_LDAP_UTL.INVALID_ROOT_ORCL_CTX</code>	Invalid Root Oracle Context.
<code>DBMS_LDAP_UTL.GENERAL_ERROR</code>	Other error.
<code>DBMS_LDAP</code> error codes	Returns proper <code>DBMS_LDAP</code> 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()`.

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-20 for more usage samples of the Property-related subprograms

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 7–41 *NORMALIZE_DN_WITH_CASE* Function Parameters

Parameter Name	Parameter Type	Parameter Description
<code>dn</code>	<code>VARCHAR2</code>	The DN.
<code>lower_case</code>	<code>PLS_INTEGER</code>	If set to 1: The normalized DN returns in lower case. If set to 0: The case is preserved in the normalized DN string.

Table 7–41 (Cont.) NORMALIZE_DN_WITH_CASE Function Parameters

Parameter Name	Parameter Type	Parameter Description
norm_dn	VARCHAR2	The normalized DN.

Return Values

Table 7–42 NORMALIZE_DN_WITH_CASE Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.GENERAL_ERROR	On failure.

Usage Notes

This function can be used while comparing two DN's.

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 7–43 *GET_PROPERTY_NAMES Function Parameters*

Parameter Name	Parameter Type	Parameter Description
pset	PROPERTY_SET	The property set in the property set collection returned from any of the following functions: DBMS_LDAP_UTL.get_group_membership(), DBMS_LDAP_UTL.get_subscriber_properties(), DBMS_LDAP_UTL.get_user_properties(), DBMS_LDAP_UTL.get_group_properties()
property_names	STRING_COLLECTION	A list of property names associated with the property set.

Return Values

Table 7–44 *GET_PROPERTY_NAMES Function Return Values*

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.PARAM_ERROR	Invalid input parameters.
DBMS_LDAP_UTL.GENERAL_ERROR	On error.

See Also

DBMS_LDAP_UTL.get_property values().

Function `get_property_values`

The function `get_property_values()` retrieves the property values (the strings) for a given property name and property.

Syntax

```
FUNCTION get_property_values
(
  pset IN PROPERTY_SET,
  property_name IN VARCHAR2,
  property_values OUT STRING_COLLECTION
)
RETURN PLS_INTEGER;
```

Parameters

Table 7–45 *GET_PROPERTY_VALUES Function Parameters*

Parameter Name	Parameter Type	Parameter Description
<code>property_name</code>	VARCHAR2	The property name.
<code>pset</code>	PROPERTY_SET	The property set in the property set collection obtained from any of the following function returns: <code>DBMS_LDAP_UTL.get_group_membership()</code> , <code>DBMS_LDAP_UTL.get_subscriber_properties()</code> , <code>DBMS_LDAP_UTL.get_user_properties()</code> , <code>DBMS_LDAP_UTL.get_group_properties()</code>
<code>property_values</code>	STRING_COLLECTION	A list of property values (strings).

Return Values

Table 7–46 *GET_PROPERTY_VALUES Function Return Values*

Value	Description
<code>DBMS_LDAP_UTL.SUCCESS</code>	On a successful completion.
<code>DBMS_LDAP_UTL.PARAM_ERROR</code>	Invalid input parameters.
<code>DBMS_LDAP_UTL.GENERAL_ERROR</code>	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 7–47 GET_PROPERTY_VALUES_LEN Function Parameters**

Parameter Name	Parameter Type	Parameter Description
property_name	VARCHAR2	A property name.
pset	PROPERTY_SET	The property set in the property set collection obtained from any of the following function returns: DBMS_LDAP_UTL.get_group_membership(), DBMS_LDAP_UTL.get_subscriber_properties(), DBMS_LDAP_UTL.get_user_properties(), DBMS_LDAP_UTL.get_group_properties()
property_values	BINVAL_COLLECTION	A list of binary property values.

Return Values**Table 7–48 GET_PROPERTY_VALUES_LEN Function Return Values**

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.

Table 7–48 (Cont.) GET_PROPERTY_VALUES_LEN Function Return Values

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().

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 7–49 FREE_PROPERTYSET_COLLECTION Procedure Parameters**

Parameter Name	Parameter Type	Parameter Description
pset_collection	PROPERTY_SET_COLLECTION	The property set collection returned from one of the following functions: DBMS_LDAP_UTL.get_group_membership(), DBMS_LDAP_UTL.get_subscriber_properties(), DBMS_LDAP_UTL.get_user_properties(), DBMS_LDAP_UTL.get_group_properties()

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 7-50 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 7-51 CREATE_MOD_PROPERTYSET Function 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 7–52 POPULATE_MOD_PROPERTYSET Function Parameters

Parameter Name	Parameter Type	Parameter Description
mod_pset	MOD_PROPERTY_SET	Mod-PropertySet data structure.
property_mod_op	PLS_INTEGER	The type of modify operation to perform on a property. 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 7–53 POPULATE_MOD_PROPERTYSET Function Return Values

Value	Description
DBMS_LDAP_UTL.SUCCESS	On a successful completion.
DBMS_LDAP_UTL.GENERAL_ERROR	Authentication failed.

Table 7–53 (Cont.) POPULATE_MOD_PROPERTYSET Function Return Values

Value	Description
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 7–54 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 7–55 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 7–56 CHECK_INTERFACE_VERSION Function Parameters

Parameter Name	Parameter Type	Parameter Description
interface_version	VARCHAR2	Version of the interface.

Return Values

Table 7–57 *CHECK_VERSION_INTERFACE Function Return Values*

Value	Description
DBMS_LDAP_ UTL.SUCCESS	Interface version is supported.
DBMS_LDAP_ UTL.GENERAL_ERROR	Interface version is not supported.

Function Return Code Summary

The DBMS_LDAP_UTL functions can return the values in the following table

Table 7–58 *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.

Table 7–58 (Cont.) Function Return Codes

Name	Return Code	Description
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.
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.

Table 7–58 (Cont.) Function Return Codes

Name	Return Code	Description
PWD_NULL_ERROR	-23	Returned by DBMS_LDAP_UTL.set_user_properties() function while changing the user password and the new user password is an empty password. This is a password policy error.
PWD_INHISTORY_ERROR	-24	Returned by DBMS_LDAP_UTL.set_user_properties() function while changing the user password and the new user password is the same as the previous password. This is a password policy error.
PWD_ILLEGALVALUE_ERROR	-25	Returned by DBMS_LDAP_UTL.set_user_properties() function while changing the user password and the new user password has an illegal character. This is a password policy error.
PWD_GRACELOGIN_WARN	-26	Returned by DBMS_LDAP_UTL.authenticate_user() function to indicate that the user password has expired and the user has been given a grace login. This is a password policy error.
PWD_MUSTCHANGE_ERROR	-27	Returned by DBMS_LDAP_UTL.authenticate_user() function when user password needs to be changed. This is a password policy error.
USER_ACCT_DISABLED_ERROR	-29	Returned by DBMS_LDAP_UTL.authenticate_user() function when user account has been disabled. This is a password policy error.
PROPERTY_NOT_FOUND	-30	Returned by user-related functions while searching for a user property in the directory.

Data-Type Summary

The DBMS_LDAP_UTL package uses the data types in the following table

Table 7–59 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.

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 Platform. These applications can be either legacy or third-party applications that are based on the Oracle platform.

This chapter contains these topics:

- [Prerequisite Knowledge](#)
- [Development Usage Model for Provisioning Integration](#)
- [Development Tasks for Provisioning Integration](#)
- [Provisioning Event Interface Description](#)

See Also: The chapter on the Oracle Directory Provisioning Integration Service in *Oracle Internet Directory Administrator's Guide*

Prerequisite Knowledge

You should be familiar with:

- Generic LDAP concepts
- Oracle Internet Directory
- Oracle Internet Directory integration with Oracle9i Application Server
- The Delegated Administration Service
- 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 Oracle9i Application Server documentation set.
- The Oracle Directory Integration Platform
- Knowledge of SQL, PL/SQL, and database RPCs

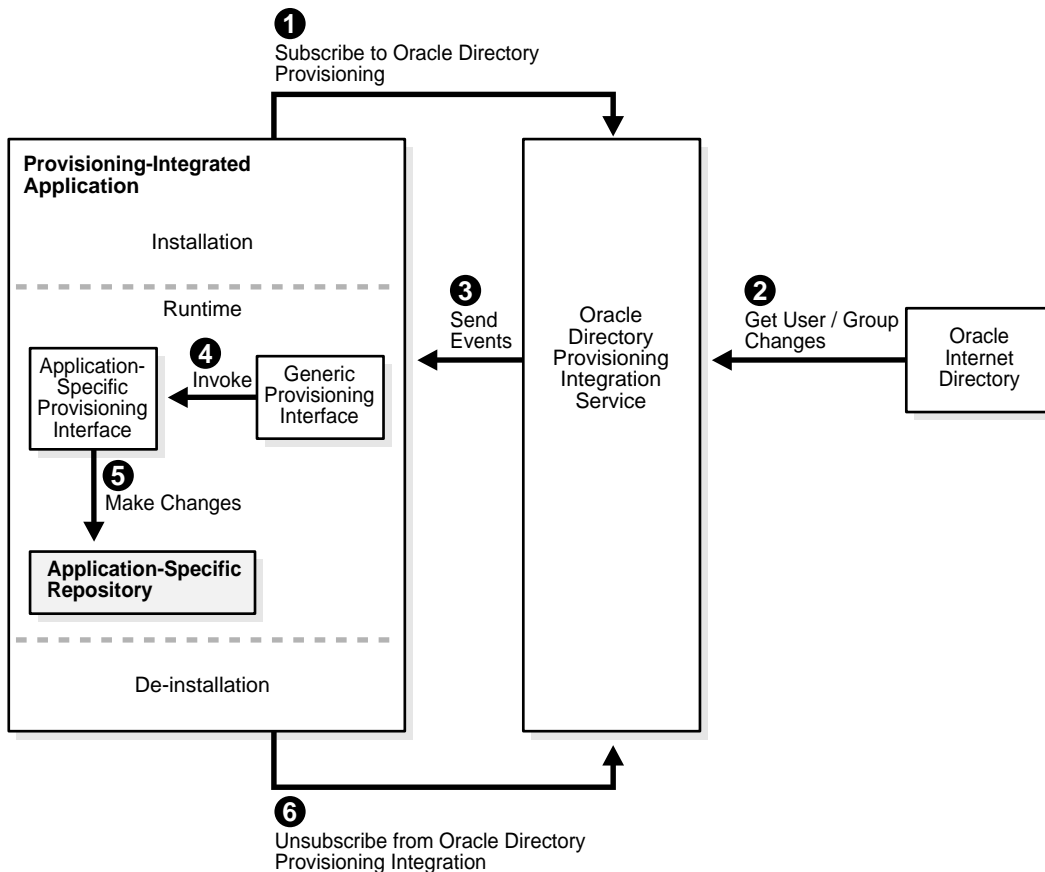
In addition, Oracle Corporation recommends that you understand Oracle9iAS Single Sign-On.

Development Usage Model for Provisioning Integration

This section gives an overview of the usage model for an agent for a provisioning-integrated application.

[Figure 8-1](#) shows the lifecycle of the application that obtains provisioning events.

Figure 8–1 How an Application Obtains Provisioning Information by Using the Oracle Directory Provisioning Integration Service



1. 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

2. The Oracle Directory Provisioning Integration Service retrieves from the Oracle Internet Directory change log the changes to user and group information. It determines which changes to send to the application.
3. The Oracle Directory Provisioning Integration Service sends the changes to the application—based on the database connect information—by invoking a generic provisioning interface.
4. The generic provisioning interface invokes the application-specific logic.
5. 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.
6. The administrator can either deinstall the application manually, or by using the automatic deinstallation process. During manual deinstallation of the application, the administrator uses the Provisioning Subscription Tool to unsubscribe the application from the provisioning platform. The Provisioning Subscription Tool is invoked from any *ORACLE_HOME* and is called `oidprovtool`.

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](#)
- [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:

- ["Provisioning Subscription Tool"](#) on page A-29 for a description of the tool's parameters and the values they can take on
- ["Development Usage Model for Provisioning Integration"](#) on page 8-2 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.

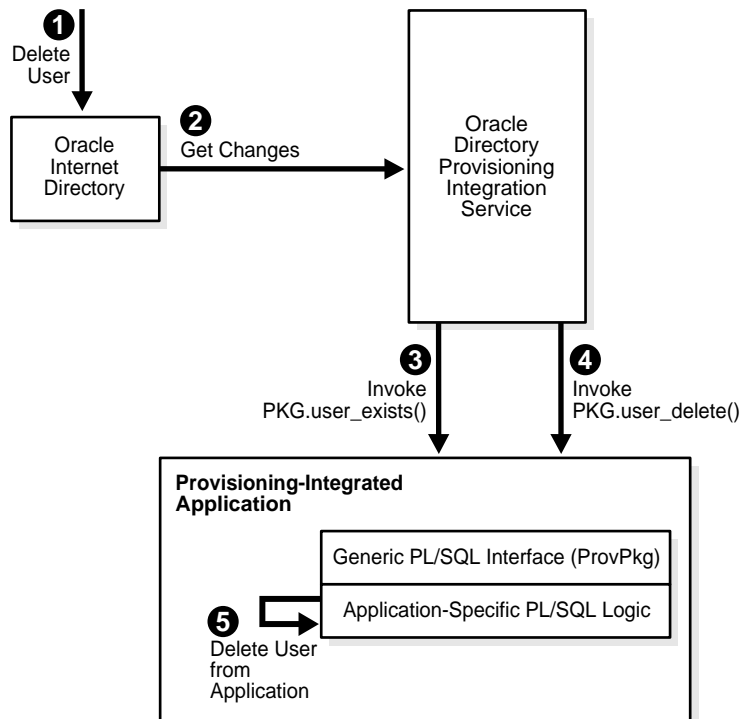
With the PL/SQL callback interface, then the application registers with the Oracle Directory Provisioning Integration Service and provides:

- 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 8–2 illustrates the system interactions for the PL/SQL callback interface.

Figure 8–2 User Deletion Using a PL/SQL Callback-Based Approach



As [Figure 8-2](#) 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.

Application Deinstallation

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

Provisioning Event Interface Description

As stated in "[Development Tasks for Provisioning Integration](#)" on page 8-4, you must develop logic to consume events generated by the Oracle Directory Provisioning Integration Service. The interface between the application and the Oracle Directory Provisioning Integration Service can be either table-based or use PL/SQL callbacks.

See Also: "[Development Usage Model for Provisioning Integration](#)" on page 8-2 for information about how to use these interfaces

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_NOTIFY
-- 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_NOTIFY AS

--
-- LDAP_NOTIFY 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;

```

LDAP_NOTIFY 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 8–1 *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 8–2 *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 doesn't 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 8–3

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

On success returns a positive number. On failure returns zero.

Oracle Internet Directory Server Plug-in Framework

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

This chapter contains these topics:

- [Introduction](#)
- [Prerequisite Knowledge](#)
- [Concepts](#)
- [Requirements](#)
- [Usage Model and Examples](#)
- [Type Definition & Usage Model](#)
- [LDAP Server Error Code Reference](#)

Introduction

The plug-in framework for Oracle Internet Directory enables developers 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` operation, they may have different ways to validate the attribute values.

Prerequisite Knowledge

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 Oracle9i Application Server
- Knowledge of SQL, PL/SQL, and database RPCs

Concepts

This section explains plug-in concepts.

This section contains these topics:

- [About Directory Server Plug-ins](#)
- [About Server Plug-in Framework](#)
- [Operation-Based Plug-ins Supported in Oracle Internet Directory](#)

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 define password policy
- 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 syntax checking and place it into Oracle Internet Directory server.
- You can provide alternate syntax/matching rules when comparing certain attribute values.

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.

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)

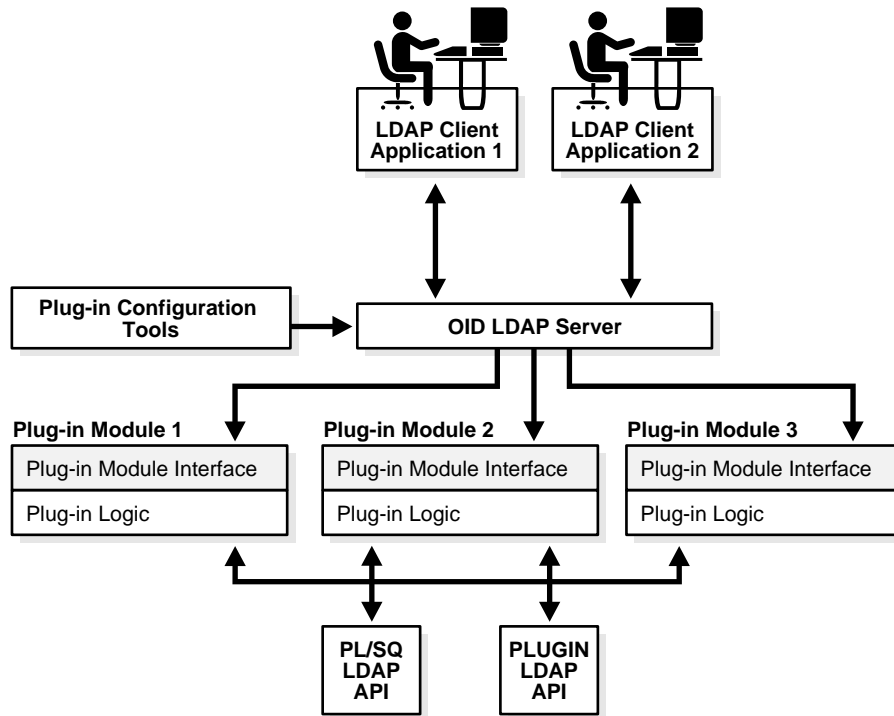
The steps to use the server plug-in framework are as follows:

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 where the following are specified:
 - Names of the plug-ins
 - Type of the plug-ins
5. Restart the LDAP server.

Figure 9–1 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 operation is executed. For example, logging and notification are post-operation plug-in functions.

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

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

When-Operation Plug-ins

The server calls when-operation plug-in modules in addition to standard 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 When Replace 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 `ldapcompare` and `ldapmodify`. When Add-on plug-ins are supported in `ldapadd`, `ldapdelete`, `ldapmodify`, `ldapcompare`, `ldapbind`, and `ldapsearch`.

Requirements

This section explains requirements for plug-ins.

This section contains these topics:

- [Designing Plug-ins](#)
- [Creating Plug-ins](#)
- [Compiling Plug-ins](#)
- [Registering Plug-ins](#)
- [Managing Plug-ins](#)
- [Enabling and Disabling Plug-ins](#)
- [Exception Handling](#)
- [Plug-in LDAP API](#)
- [Plug-in and Replication](#)
- [Plug-in and DB Tools](#)
- [Security](#)

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 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 9–1 *Plug-in Module Interface*

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

Table 9–1 (Cont.) Plug-in Module Interface

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

See Also: [Plug-in Module Interface Specifications](#) on page 9-24 and [Usage Model and Examples](#) on page 9-18 for coding examples

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

Table 9–2 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	WHEN_BIND	Ldapcontext, Bind DN, Password	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	WHEN_COMPARE	Ldapcontext, DN, attribute, value	return code, error message

Table 9–2 (Cont.) Operation-Based and Attribute-Based Plug-in Procedure

Invocation Context	Procedure Name	IN Parameters	OUT Parameters
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
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
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
With ldapsearch	WHEN_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 9-14 for valid values for the return code and error message
- [LDAP Server Error Code Reference](#) on page 9-28 for valid values for the OUT parameters return code
- [Plug-in Module Interface Specifications](#) on page 9-24 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 9-3](#) lists and describes its attributes.

Table 9-3 Plug-in Attribute Names and Values

Attribute Name	Attribute Value	Mandatory	Optional
Cn	Plug-in entry name	X	-
orclPluginName	Plug-in package name	X	-
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 9-4	X	-
orclPluginKind	PL/SQL	-	X
orclPluginEnable	0 = disable (default) 1 = enable	-	X
orclPluginVersion	Supported plug-in version number	-	X
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.	-	X
orclPluginLDAPOperation	One of the following values: ldapcompare ldapmodify ldapbind ldapadd ldapdelete ldapsearch	-	X

Table 9–3 (Cont.) Plug-in Attribute Names and Values

Attribute Name	Attribute Value	Mandatory	Optional
orclPluginTiming	One of the following values: pre when post	-	X
orclPluginIsReplace	0 = disable (default) 1 = enable For WHEN timing plug-in only	-	X
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.	-	X

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
```

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
```

When you have added this entry to the directory, the directory server validates the plug-in by quickly executing it and checking for compilation or access privilege errors. It then gathers more information about this plug-in—such as timing and the type of LDAP operation related to the plug-in.

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 stored procedures.

Enabling and Disabling Plug-ins

To turn the plug-in on/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=subconfigsentry` to be 1/0. You must restart the Oracle Internet Directory server after you modify the `orclPluginEnable` value.

Exception Handling

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

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 LDAP Server Error Code Reference on page 9-28
-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 9–4 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_COMPARE, WHEN_DELETE	LDAP operation is completed. There is no rollback.
WHEN_BIND, WHEN_MODIFY, WHEN_ADD, WHEN_SEARCH, WHEN_COMPARE, WHEN_DELETE	Rollback the LDAP operation

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

Table 9–5 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_COMPARE, WHEN_DELETE	Pre-operation plug-in is completed. There is no rollback.
POST_BIND, POST_MODIFY, POST_ADD, POST_SEARCH, WHEN_COMPARE, WHEN_DELETE	Proceed with post-operation plug-in. The LDAP operation result is one of the IN parameters.
WHEN_BIND, WHEN_MODIFY, WHEN_ADD, WHEN_SEARCH, WHEN_COMPARE, 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.

See Also: The DBMS_LDAP User's Guide for information about use of the LDAP PL/SQL API

- 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 LDAP 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 Oracle Internet 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 9-18 for coding examples

Plug-in and Replication

There are cases that can cause an inconsistent state in a replication environment as follows:

- Plug-in metadata replicated to other nodes
- `Ldapmodify`, `ldapadd`, or any other LDAP operation that will change the entries in the directory are used in the plug-in program
- Only some of the participating nodes install the plug-in
- The plug-in implements extra checking that depends on the directory data

Plug-in and DB 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 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;
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;
  errmsg := 'no post_search plguin error msg';
  COMMIT;
EXCEPTION
  WHEN others THEN
    rc := 1;
    errmsg := '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
```

6. Restart the Oracle Internet Directory server

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 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
  (ldapplugincontext IN ODS.plugincontext,
  result IN INTEGER,
  dn IN VARCHAR2,
  rc OUT INTEGER,
```

```
        errmsg OUT VARCHAR2
    );
END LDAP_PLUGIN_EXAMPLE2;
/
```

3. Create plug-in package body.

```
CREATE OR REPLACE PACKAGE BODY LDAP_PLUGIN_EXAMPLE2 AS
PROCEDURE post_delete
    (ldapplugincontext IN ODS.plugincontext,
    result IN INTEGER,
    dn IN VARCHAR2,
    rc OUT INTEGER,
    errmsg 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;
    errmsg := 'no post_delete plugin error msg';
EXCEPTION
    WHEN others THEN
        rc := 1;
        errmsg := 'exception: post_delete plugin';
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

```

5. Restart the Oracle Internet Directory server

Type Definition & Usage Model

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

This section contains these topics:

- [Database Object Type Definitions](#)
- [Plug-in Module Interface Specifications](#)

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,
errmsg     OUT VARCHAR2
```



```
);

PROCEDURE when_add (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,
    errmsg  OUT VARCHAR2
);

PROCEDURE when_compare (ldapplugincontext IN ODS.plugincontext,
    dn      IN VARCHAR2,
    attrname IN VARCHAR2,
    attrval IN VARCHAR2,
    rc      OUT INTEGER,
    errmsg  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,
    errmsg  OUT VARCHAR2
);

PROCEDURE post_compare (ldapplugincontext IN ODS.plugincontext,
    result  IN INTEGER,
    dn      IN VARCHAR2,
    attrname IN VARCHAR2,
    attrval IN VARCHAR2,
    rc      OUT INTEGER,
    errmsg  OUT VARCHAR2
);

PROCEDURE pre_delete (ldapplugincontext IN ODS.plugincontext,
    dn      IN VARCHAR2,
    rc      OUT INTEGER,
    errmsg  OUT VARCHAR2
);

PROCEDURE when_delete (ldapplugincontext IN ODS.plugincontext,
    dn      IN VARCHAR2,
    rc      OUT INTEGER,
    errmsg  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 when_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 (ldapplugincontext IN ODS.plugincontext,
```

```

        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;
/

```

LDAP 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_CRITICAL                   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;

}

Part III

Appendixes

Part III explains the command-line tools, including generic tools and Oracle-specific tools. It contains these appendixes:

- [Appendix A, "Command-Line Tools Syntax"](#)
- [Appendix B, "Sample Usage"](#)

Command-Line Tools Syntax

This chapter provides syntax, usage notes, and examples for using LDAP Data Interchange Format (LDIF) and LDAP command-line tools. It contains these topics:

- [LDAP Data Interchange Format \(LDIF\) Syntax](#)
- [Entry-Management Command-Line Tools](#)
- [Attribute-Management Command-Line Tools](#)
- [Provisioning Subscription Tool](#)

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:	<i>RDN,RDN,RDN, ...</i>	Separate RDNs with commas.
<i>attribute</i> :	<i>attribute_value</i>	This line repeats for every attribute in the entry, and for every attribute value in multi-valued attributes.
objectClass:	<i>object_class_value</i>	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
email: ssmith@us.Acme.com
telephoneNumber: 69332
photo:/ORACLE_HOME/empdir/photog/ssmith.jpg
objectClass: organizational person
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 *Oracle Internet Directory Administrator's Guide*, for a complete list of LDIF formatting rules and for information about using Globalization Support with LDIF files.

Entry-Management Command-Line Tools

This section tells you how to use the following tools:

- [ldapadd Syntax](#)
- [ldapaddmt Syntax](#)
- [ldapbind Syntax](#)
- [ldapdelete Syntax](#)
- [ldapmoddn Syntax](#)
- [ldapsearch Syntax](#)

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-22.

See Also: *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 filename
```

where *filename* 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
```

Optional Arguments	Descriptions
-b	Specifies that you have included binary file names in the file, which are preceded by a forward slash character. The tool retrieves the actual values from the file referenced.
-c	Tells ldapadd to proceed in spite of errors. The errors will be reported. (If you do not use this option, ldapadd stops when it encounters an error.)
-D <i>binddn</i>	When authenticating to the directory, specifies doing so as the entry specified in <i>binddn</i> . Use this with the <i>-w password</i> option.
-E " <i>character_set</i> "	Specifies native character set encoding. See the chapter on Globalization Support in <i>Oracle Internet Directory Administrator's Guide</i> .
-f <i>filename</i>	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 <i>ldaphost</i>	Connects to <i>ldaphost</i> , rather than to the default host, that is, your local computer. <i>ldaphost</i> can be a computer name or an IP address.
-K	Same as <i>-k</i> , 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 <i>ref_hop_limit</i>	Specifies the number of referral hops that a client should process. The default value is 5.
-p <i>directory_server_port_number</i>	Connects to the directory on TCP port <i>directory_server_port_number</i> . If you do not specify this option, the tool connects to the default port (389).
-P <i>wallet_password</i>	Specifies wallet password required for one-way or two-way SSL connections

Optional Arguments	Descriptions
<code>-U <i>SSLAuth</i></code>	Specifies SSL authentication mode: <ul style="list-style-type: none">▪ 1 for no authentication required▪ 2 for one way authentication required▪ 3 for two way authentication required
<code>-v</code>	Specifies verbose mode
<code>-V <i>ldap_version</i></code>	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.
<code>-w <i>password</i></code>	Provides the password required to connect
<code>-W <i>wallet_location</i></code>	Specifies wallet location required for one-way or two-way SSL connections. For example, on Solaris, you could set this parameter as follows: <pre>-W "file:/home/my_dir/my_wallet"</pre> On Windows NT, you could set this parameter as follows: <pre>-W "file:C:\my_dir\my_wallet"</pre>

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 filename
```

where *filename* 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.

Optional Arguments	Descriptions
-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 <i>binddn</i>	When authenticating to the directory, specifies doing so as the entry is specified in <i>binddn</i> . Use this with the <i>-w password</i> option.
-E " <i>character_set</i> "	Specifies native character set encoding. See the chapter on Globalization Support in <i>Oracle Internet Directory Administrator's Guide</i> .
-h <i>ldaphost</i>	Connects to <i>ldaphost</i> , rather than to the default host, that is, your local computer. <i>ldaphost</i> can be a computer name or an IP address.

Optional Arguments	Descriptions
-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 <i>ref_hop_limit</i>	Specifies the number of referral hops that a client should process. The default value is 5.
-p <i>ldapport</i>	Connects to the directory on TCP port <i>ldapport</i> . If you do not specify this option, the tool connects to the default port (389).
-P <i>wallet_password</i>	Specifies wallet password required for one-way or two-way SSL connections
-T	Sets the number of threads for concurrently processing entries
-U <i>SSLAuth</i>	Specifies SSL Authentication Mode: <ul style="list-style-type: none">■ 1 for no authentication required■ 2 for one way authentication required■ 3 for two way authentication required
-v	Specifies verbose mode
-V <i>ldap_version</i>	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 <i>password</i>	Provides the password required to connect
-W <i>wallet_location</i>	Specifies wallet location required for one-way or two-way SSL connections. For example, on Solaris, you could set this parameter as follows: <code>-W "file:/home/my_dir/my_wallet"</code> On Windows NT, you could set this parameter as follows: <code>-W "file:C:\my_dir\my_wallet"</code>

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]
```

Optional Arguments	Descriptions
-D <i>binddn</i>	When authenticating to the directory, specifies doing so as the entry specified in <i>binddn</i> . Use this with the <i>-w password</i> option.
-E ". <i>character_set</i> "	Specifies native character set encoding. See the chapter on Globalization Support in <i>Oracle Internet Directory Administrator's Guide</i> .
-h <i>ldaphost</i>	Connects to <i>ldaphost</i> , rather than to the default host, that is, your local computer. <i>ldaphost</i> can be a computer name or an IP address.
-n	Shows what would occur without actually performing the operation
-p <i>ldapport</i>	Connects to the directory on TCP port <i>ldapport</i> . If you do not specify this option, the tool connects to the default port (389).
-P <i>wallet_password</i>	Specifies the wallet password required for one-way or two-way SSL connections
-U <i>SSLAuth</i>	Specifies SSL authentication mode: <ul style="list-style-type: none"> ■ 1 for no authentication required ■ 2 for one way authentication required ■ 3 for two way authentication required
-V <i>ldap_version</i>	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 <i>password</i>	Provides the password required to connect
-W <i>wallet_location</i>	Specifies wallet location required for one-way or two-way SSL connections. For example, on Solaris, you could set this parameter as follows: <pre>-W "file:/home/my_dir/my_wallet"</pre> <p>On Windows NT, you could set this parameter as follows:</p> <pre>-W "file:C:\my_dir\my_wallet"</pre>

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_filename]
```

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"
```

Optional Arguments	Descriptions
<i>-D binddn</i>	When authenticating to the directory, uses a full DN for the <i>binddn</i> parameter; typically used with the <i>-w password</i> option.
<i>-d debug-level</i>	Sets the debugging level. See the chapter on managing a directory server in <i>Oracle Internet Directory Administrator's Guide</i> .
<i>-E "character_set"</i>	Specifies native character set encoding. See the chapter on Globalization Support in <i>Oracle Internet Directory Administrator's Guide</i> .
<i>-f input_filename</i>	Specifies the input filename
<i>-h ldaphost</i>	Connects to <i>ldaphost</i> , rather than to the default host, that is, your local computer. <i>ldaphost</i> can be a computer name or an IP address.
<i>-k</i>	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.
<i>-M</i>	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.
<i>-n</i>	Shows what would be done, but doesn't actually delete
<i>-O ref_hop_limit</i>	Specifies the number of referral hops that a client should process. The default value is 5.
<i>-p ldapport</i>	Connects to the directory on TCP port <i>ldapport</i> . If you do not specify this option, the tool connects to the default port (389).

Optional Arguments	Descriptions
-P <i>wallet_password</i>	Specifies wallet password required for one-way or two-way SSL connections
-U <i>SSLAuth</i>	Specifies SSL authentication mode: <ul style="list-style-type: none"> ■ 1 for no authentication required ■ 2 for one way authentication required ■ 3 for two way authentication required
-v	Specifies verbose mode
-V <i>ldap_version</i>	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 <i>password</i>	Provides the password required to connect.
-W <i>wallet_location</i>	Specifies wallet location required for one-way or two-way SSL connections. For example, on Solaris, you could set this parameter as follows: <pre style="margin-left: 40px;">-W "file:/home/my_dir/my_wallet"</pre> <p>On Windows NT, you could set this parameter as follows:</p> <pre style="margin-left: 40px;">-W "file:C:\my_dir\my_wallet"</pre>

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=dcpl" to "cn=thanh mai". It uses port 389, and a host named myhost.

```
ldapmoddn -p 389 -h myhost -b "cn=dcpl,dc=Americas,dc=imc,dc=com" -R "cn=thanh mai"
```

Mandatory Argument	Description
-b " <i>basedn</i> "	Specifies DN of the entry to be moved
Optional Arguments	Descriptions
-D <i>binddn</i>	When authenticating to the directory, do so as the entry is specified in <i>binddn</i> . Use this with the <i>-w password</i> option.
-E " <i>character_set</i> "	Specifies native character set encoding. See the chapter on Globalization Support in <i>Oracle Internet Directory Administrator's Guide</i> .
-f <i>filename</i>	Specifies the input filename
-h <i>ldaphost</i>	Connects to <i>ldaphost</i> , rather than to the default host, that is, your local computer. <i>ldaphost</i> can be a computer name or an IP address.
-M	Instructs the tool to send the <code>ManageDSAIT</code> control to the server. The <code>ManageDSAIT</code> control instructs the server not to send referrals to clients. Instead a referral entry is returned as a regular entry.
-N <i>newparent</i>	Specifies new parent of the RDN
-O <i>ref_hop_limit</i>	Specifies the number of referral hops that a client should process. The default value is 5.
-p <i>ldapport</i>	Connects to the directory on TCP port <i>ldapport</i> . If you do not specify this option, the tool connects to the default port (389).
-P <i>wallet_password</i>	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 <i>newrdn</i>	Specifies new RDN
-U <i>SSLAuth</i>	Specifies SSL authentication mode: <ul style="list-style-type: none"> ■ 1 for no authentication required ■ 2 for one way authentication required ■ 3 for two way authentication required
-V <i>ldap_version</i>	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.

Optional Arguments	Descriptions
<code>-w password</code>	Provides the password required to connect.
<code>-W wallet_location</code>	Specifies wallet location required for one-way or two-way SSL connections. For example, on Solaris, you could set this parameter as follows: <pre>-W "file:/home/my_dir/my_wallet"</pre> On Windows NT, you could set this parameter as follows: <pre>-W "file:C:\my_dir\my_wallet"</pre>

ldapsearch Syntax

The ldapsearch command-line tool enables you to search for and retrieve specific entries in the directory.

ldapsearch uses this syntax:

```
ldapsearch [arguments] filter [attributes]
```

The *filter* format must be compliant with RFC-2254.

See Also: <http://www.ietf.org/rfc/rfc2254.txt> 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.

Mandatory Arguments	Descriptions
<code>-b "basedn"</code>	Specifies the base DN for the search
<code>-s scope</code>	Specifies search scope: base, one, or sub.

Optional Arguments	Descriptions
<code>-A</code>	Retrieves attribute names only (no values)
<code>-a deref</code>	Specifies alias dereferencing: never, always, search, or find
<code>-B</code>	Allows printing of non-ASCII values
<code>-D binddn</code>	When authenticating to the directory, specifies doing so as the entry specified in <i>binddn</i> . Use this with the <code>-w password</code> option.

Optional Arguments	Descriptions
-d <i>debug level</i>	Sets debugging level to the level specified (see the chapter on managing a directory server in <i>Oracle Internet Directory Administrator's Guide</i> .)
-E " <i>character_set</i> "	Specifies native character set encoding. See the chapter on Globalization Support in <i>Oracle Internet Directory Administrator's Guide</i> .
-f <i>file</i>	Performs sequence of searches listed in <i>file</i>
-F <i>sep</i>	Prints ' <i>sep</i> ' instead of '=' between attribute names and values
-h <i>ldaphost</i>	Connects to <i>ldaphost</i> , rather than to the default host, that is, your local computer. <i>ldaphost</i> can be a computer name or an IP address.
-L	Prints entries in LDIF format (-B is implied)
-l <i>timelimit</i>	Specifies maximum time (in seconds) to wait for <i>ldapsearch</i> command to complete
-M	Instructs the tool to send the <i>ManageDSAIT</i> control to the server. The <i>ManageDSAIT</i> control instructs the server not to send referrals to clients. Instead a referral entry is returned as a regular entry.
-n	Shows what would be done without actually searching
-O <i>ref_hop_limit</i>	Specifies the number of referral hops that a client should process. The default value is 5.
-p <i>ldapport</i>	Connects to the directory on TCP port <i>ldapport</i> . If you do not specify this option, the tool connects to the default port (389).
-P <i>wallet_password</i>	Specifies wallet password (required for one-way or two-way SSL connections)
-S <i>attr</i>	Sorts the results by attribute <i>attr</i>
-t	Writes to files in <i>/tmp</i>
-u	Includes user friendly entry names in the output
-U <i>SSLAuth</i>	Specifies the SSL authentication mode: <ul style="list-style-type: none"> ■ 1 for no authentication required ■ 2 for one way authentication required ■ 3 for two way authentication required
-v	Specifies verbose mode

Optional Arguments	Descriptions
<code>-V ldap_version</code>	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.
<code>-w passwd</code>	Specifies bind passwd for simple authentication
<code>-W wallet_location</code>	Specifies wallet location required for one-way or two-way SSL connections. For example, on Solaris, you could set this parameter as follows: <pre>-W "file:/home/my_dir/my_wallet"</pre> On Windows NT, you could set this parameter as follows: <pre>-W "file:C:\my_dir\my_wallet"</pre>
<code>-z sizelimit</code>	Specifies maximum number of entries to retrieve

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 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 sub-tree search and returns all entries having a DN starting with `"cn=Person"`.

```
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 (dn) 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 `orcle` 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=orcle*"
```

The following example searches for entries where the `objectclass` attribute begins with `orcle` and `cn` begins with `foo`.

```
ldapsearch -p 389 -h sun1 -b "ou=hr, o=acme, c=us" -s subtree "(&(objectclass=orcle*)(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"
```

Attribute-Management Command-Line Tools

This section contains these topics:

- [The Catalog Management Tool](#)
- [ldapcompare Syntax](#)
- [ldapmodify Syntax](#)
- [ldapmodifymt Syntax](#)

The Catalog Management Tool

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
- Less than 28 characters in their names

See Also: *Oracle Internet Directory Administrator's Guide* for the matching rules supported by Oracle Internet Directory

If you want to use additional attributes in search filters, 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.

Note: To run shell script tools on the Windows operating system, you need one of the following UNIX emulation utilities:

- Cygwin 1.0. Visit: <http://sources.redhat.com/cygwin/>
 - MKS Toolkit 5.1 or 6.0. Visit: <http://www.datafocus.com/products/>
-
-

The Catalog Management tool uses this syntax:

```
catalog.sh -connect net_service_name {add|delete} {-attr attr_name|-file filename}
```

Mandatory Argument	Description
<code>-connect net_ service_name</code>	Specifies the net service name to connect to the directory database See Also: <i>Oracle9i Net Services Administrator's Guide</i> in the Oracle Database Documentation Library

Optional Argument	Description
<code>- add -attr attr_ name</code>	Indexes the specified attribute
<code>- delete -attr attr_name</code>	Drops the index from the specified attribute
<code>- add -file filename</code>	Indexes attributes (one for each line) in the specified file
<code>-delete -file filename</code>	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 chapter on preliminary tasks in *Oracle Internet Directory Administrator's Guide*, for instructions on starting and restarting directory servers

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
```

Mandatory Arguments	Descriptions
-a <i>attribute name</i>	Specifies the attribute on which to perform the compare
-b " <i>basedn</i> "	Specifies the distinguished name of the entry on which to perform the compare
-v <i>attribute value</i>	Specifies the attribute value to compare

Optional Arguments	Descriptions
-D <i>binddn</i>	When authenticating to the directory, specifies doing so as the entry is specified in <i>binddn</i> . Use this with the <i>-w password</i> option.
-d <i>debug-level</i>	Sets the debugging level. See the chapter on managing a directory server in <i>Oracle Internet Directory Administrator's Guide</i> .
-E " <i>character_set</i> "	Specifies native character set encoding. See chapter on Globalization Support in <i>Oracle Internet Directory Administrator's Guide</i> .
-f <i>filename</i>	Specifies the input filename
-h <i>ldaphost</i>	Connects to <i>ldaphost</i> , rather than to the default host, that is, your local computer. <i>ldaphost</i> 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 <i>ref_hop_limit</i>	Specifies the number of referral hops that a client should process. The default value is 5.
-p <i>ldapport</i>	Connects to the directory on TCP port <i>ldapport</i> . If you do not specify this option, the tool connects to the default port (389).
-P <i>wallet_password</i>	Specifies wallet password (required for one-way or two-way SSL connections)

Optional Arguments	Descriptions
<i>-U SSLAuth</i>	Specifies SSL authentication mode: <ul style="list-style-type: none">■ 1 for no authentication required■ 2 for one way authentication required■ 3 for two way authentication required
<i>-V ldap_version</i>	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.
<i>-w password</i>	Provides the password required to connect
<i>-W wallet_location</i>	Specifies wallet location required for one-way or two-way SSL connections. For example, on Solaris, you could set this parameter as follows: <code>-W "file:/home/my_dir/my_wallet"</code> On Windows NT, you could set this parameter as follows: <code>-W "file:C:\my_dir\my_wallet"</code>

ldapmodify Syntax

The ldapmodify tool enables you to act on attributes.

ldapmodify uses this syntax:

```
ldapmodify [arguments] -f filename
```

where *filename* 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.

Optional Arguments	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 <i>binddn</i>	When authenticating to the directory, specifies doing so as the entry is specified in <i>binddn</i> . Use this with the <i>-w password</i> option.
-E " <i>character_set</i> "	Specifies native character set encoding. See the chapter on Globalization Support in <i>Oracle Internet Directory Administrator's Guide</i> .
-h <i>ldaphost</i>	Connects to <i>ldaphost</i> , rather than to the default host, that is, your local computer. <i>ldaphost</i> 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 <i>log_file_name</i>	Can be used with the <i>-c</i> option to write the erroneous LDIF entries in the logfile. You must specify the absolute path for the log file name.
-O <i>ref_hop_limit</i>	Specifies the number of referral hops that a client should process. The default value is 5.
-p <i>ldapport</i>	Connects to the directory on TCP port <i>ldapport</i> . If you do not specify this option, the tool connects to the default port (389).

Optional Arguments	Description
<code>-P wallet_password</code>	Specifies wallet password required for one-way or two-way SSL connections
<code>-U SSLAuth</code>	Specifies SSL authentication mode: <ul style="list-style-type: none"> ■ 1 for no authentication required ■ 2 for one way authentication required ■ 3 for two way authentication required
<code>-v</code>	Specifies verbose mode
<code>-V ldap_version</code>	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.
<code>-w password</code>	Overrides the default, unauthenticated, null bind. To force authentication, use this option with the <code>-D</code> option.
<code>-W wallet_location</code>	Specifies wallet location required for one-way or two-way SSL connections. For example, on Solaris, you could set this parameter as follows: <pre style="margin-left: 40px;">-W "file:/home/my_dir/my_wallet"</pre> <p>On Windows NT, you could set this parameter as follows:</p> <pre style="margin-left: 40px;">-W "file:C:\my_dir\my_wallet"</pre>

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 below:

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 Fritchey,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 Fritchey,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 in Line 1 above. 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 Fritchey,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 Fritchey,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 Fritchey,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 Fritchey,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 Fritch,y,ou=Sales,o=Oracle,c=US
changetype:modrdn
newrdn: cn=Barbara Fritch-Blomberg
```

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 filename
```

where *filename* 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-22 for additional formatting specifications used by `ldapmodifymt`

For example:

```
ldapmodifymt -T 5 -h node1 -p 3000 -f myentries.ldif
```

Optional Arguments	Descriptions
-a	Denotes that entries are to be added, and that the input file is in LDIF format. (If you are running <code>ldapadd</code> , this flag is not required.)
-b	Specifies that you have included binary file names in the data file, which are preceded by a forward slash character.
-c	Tells <code>ldapmodify</code> to proceed in spite of errors. The errors will be reported. (If you do not use this option, <code>ldapmodify</code> stops when it encounters an error.)
-D " <i>binddn</i> "	When authenticating to the directory, specifies doing so as the entry is specified in <code>binddn</code> . Use this with the <code>-w <i>password</i></code> option.
-E " <i>character_set</i> "	Specifies native character set encoding. See the chapter on Globalization Support in <i>Oracle Internet Directory Administrator's Guide</i> .

Optional Arguments	Descriptions
-h <i>ldaphost</i>	Connects to <i>ldaphost</i> , rather than to the default host, that is, your local computer. <i>ldaphost</i> 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 <i>ref_hop_limit</i>	Specifies the number of referral hops that a client should process. The default value is 5.
-p <i>ldapport</i>	Connects to the directory on TCP port <i>ldapport</i> . If you do not specify this option, the tool connects to the default port (389).
-P <i>wallet_password</i>	Specifies wallet password required for one-way or two-way SSL connections
-T	Sets the number of threads for concurrently processing entries
-U <i>SSLAuth</i>	Specifies SSL authentication mode: <ul style="list-style-type: none"> ■ 1 for no authentication required ■ 2 for one way authentication required ■ 3 for two way authentication required
-v	Specifies verbose mode
-V <i>ldap_version</i>	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 <i>password</i>	Overrides the default, unauthenticated, null bind. To force authentication, use this option with the -D option.
-W <i>wallet_location</i>	Specifies wallet location required for one-way or two-way SSL connections. For example, on Solaris, you could set this parameter as follows: <pre>-W "file:/home/my_dir/my_wallet"</pre> <p>On Windows NT, you could set this parameter as follows:</p> <pre>-W "file:C:\my_dir\my_wallet"</pre>

Provisioning Subscription Tool

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 DIP 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 only be one provisioning profile per application per subscriber.

Note: To run shell script tools on the Windows operating system, you need one of the following UNIX emulation utilities:

- Cygwin 1.0. Visit: <http://sources.redhat.com/cygwin/>
 - MKS Toolkit 5.1 or 6.0. Visit: <http://www.datafocus.com/products/>
-
-

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-1

Name	Operations	Mandatory/Optional	Description
operation	all	M	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 per invocation of the tool.
ldap_host	all	O	Host-name of the LDAP server on which the subscription operations are to be performed. If not specified, the default value of 'localhost' is assumed.
ldap_port	all	O	The TCP/IP port on which the LDAP server is listening for requests. If not specified, the default value of '389' is assumed.
ldap_user_dn	all	M	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.
ldap_user_password	all	M	The password of the user on whose behalf the operation is to be performed.
application_dn	all	M	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.
organization_dn	all	M	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.

Table A-1

Name	Operations	Mandatory/Optional	Description
interface_name	create only	M	Database schema name for the PLSQL package. Format of the value should be: [Schema].[PACKAGE_NAME]
interface_type	create only	O	The type of the interface to which events have to be propagated. Valid Values: PLSQL (if not specified this is assumed as the default)
interface_connect_info	create only	M	Database connect string Format of this string: [HOST]:[PORT]:[SID]:[USER_ID]:[PASSWORD]
interface_version	create only	O	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.
interface_additional_info	create only	O	Additional information for the interface. This is not currently used.
schedule	create only	O	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.
max_retries	create only	O	The number of times the Provisioning Service should retry a failed event delivery. If not specified, a default value of 5 is assumed.
event_subscription	create only	O	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(i.e. send user and group delete notifications under the organization DN).

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](#)
- [Using DBMS_LDAP for a Search](#)

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

Table 9–6 Employee Details(Columns) in Database Table(EMP)

Column	Datatype
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 :

Table 9–7 Corresponding Data Representation in LDAP Directory

LDAP Representation	Database Table Representation
DN	cn=FIRST_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) constitute 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 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

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);
  -- DEMS_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](#)
- [Example: Property-Related Subprograms](#)
- [Example: Subscriber-Related Functions](#)
- [Example: Group-Related Functions](#)

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 an 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 an 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 an 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](#)
- [Subscriber Class Sample Code](#)
- [Group Class Sample Code](#)
- [Print Sample Code](#)

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,
                                 "93B37BEC3B1F46F8E034080020F73460",
                                 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 PropertySetCollection 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 systematic 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
```

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 Replication](#)

agent

See [directory integration agent](#)

agent profile

In an Oracle Directory Integration Platform environment, an entry in Oracle Internet Directory that specifies:

- Configuration parameters for integration agents
- Mapping rules for synchronizing between a connected directory and Oracle Internet Directory

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 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 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 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.

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 run-time. The configuration set entries are maintained in the subtree specified by the subConfigsubEntry attribute of the DSE, which itself resides in the associated **directory information base (DIB)** against which the servers are started.

connect descriptor

A specially formatted description of the destination for a network connection. A connect descriptor contains destination service and network route information.

The destination service is indicated by using its service name for Oracle9i release 9.2 database or its Oracle System Identifier (SID) for Oracle release 8.0 or version 7 databases. The network route provides, at a minimum, the location of the listener through use of a network address.

connected directory

In an Oracle Directory Integration Platform environment, an information repository requiring full synchronization of data between Oracle Internet Directory and itself—for example, an Oracle human Resources database.

consumer

A directory server that is the destination of replication updates. Sometimes called a slave.

contention

Competition for resources.

context prefix

The **DN** of the root of a **naming context**.

cryptography

The practice of encoding and decoding data, resulting in secure messages.

data integrity

The guarantee that the contents of the message received were not altered from the contents of the original message sent.

decryption

The process of converting the contents of an encrypted message (ciphertext) back into its original readable format (plaintext).

default knowledge reference

A **knowledge reference** that is returned when the base object is not in the directory, and the operation is performed in a naming context not held locally by the server. A default knowledge reference typically sends the user to a server that has more knowledge about the directory partitioning arrangement.

default subscriber

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 subscriber, and the enterprises that are hosted are called subscribers.

Delegated Administration Service

A set of individual, pre-defined services—called Delegated Administration Service units—for performing directory operations on behalf of a user. It makes it easier to develop and deploy administration solutions for both Oracle directory-enabled applications and other directory-enabled applications that use Oracle Internet Directory.

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 subscriber domains, 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 agent

In an Oracle Directory Integration Platform environment, a program that interacts with a connected directory to synchronize changes between the connected directory and Oracle Internet Directory.

directory integration profile

In an Oracle Directory Integration Platform environment, an entry in Oracle Internet Directory that describes how Oracle Directory Integration Platform communicates with external systems and what is communicated.

directory integration 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 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 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 server. The Oracle directory integration 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.

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)

In a multi-master replication environment, an entry replicated on multiple nodes has the same DN on each node. However, even though it has the same DN, it is assigned a different GUID on each node. For example, the same DN can be replicated on both node1 and node2, but the GUID for that DN as it resides on node1 would be different from the GUID for that DN on node2.

grace login

A login occurring within the specified period before password expiration.

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.

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](#).

integration agent

See [agent](#).

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.

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](#)

native agent

In an Oracle Directory Integration Platform environment, an **agent** that runs under the control of the [directory integration server](#).

naming attribute

A specialized attribute that holds values for different types of **RDN**. A naming attribute is identifiable by its mnemonic label, usually `cn`, `sn`, `ou`, `o`, `c`, and so on. For example, the naming attribute `c` is the mnemonic for the naming attribute `country`, and it holds the RDN for specific country values.

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 references](#) (also called referrals) to subordinate naming contexts. It can range in size from a single entry to the entire DIT.

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

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 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 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 server (DIS)

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 Internet Directory

A general purpose directory service that enables retrieval of information about dispersed users and network resources. It combines Lightweight Directory Access Protocol (LDAP) Version 3 with the high performance, scalability, robustness, and availability of Oracle9i.

Oracle Net Services

The foundation of the Oracle family of networking products, allowing services and their client applications to reside on different computers and communicate. The main function of Oracle Net Services is to establish network sessions and transfer data between a client application and a server. Oracle Net Services is located on each computer in the network. Once a network session is established, Oracle Net Services acts as a data courier for the client and the server.

Oracle PKI certificate usages

Defines Oracle application types that a **certificate** supports.

Oracle Wallet Manager

A Java-based application that security administrators use to manage public-key security credentials on clients and servers.

See Also: *Oracle Advanced Security Administrator's Guide*

Oracle9i Replication

A feature in Oracle9i that enables database tables to be kept synchronized across two Oracle databases.

other information repository

In an Oracle Directory Integration 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.

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.

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.

referral

Information that a directory server provides to a client and which points to other servers the client must contact to find the information it is requesting.

See also [knowledge reference](#).

relational database

A structured collection of data that stores data in tables consisting of one or more rows, each containing the same set of columns. Oracle makes it very easy to link the data in multiple tables. This is what makes Oracle a relational database management system, or RDBMS. It stores data in two or more tables and enables you to define relationships between the tables. The link is based on one or more fields common to both tables.

replica

Each copy of a naming context that is contained within a single server.

RDN

See [relative distinguished name \(RDN\)](#).

registry entry

An entry containing runtime information associated with invocations of Oracle directory servers, called a [directory server instance](#). Registry entries are stored in the directory itself, and remain there until the corresponding directory server instance stops.

relative distinguished name (RDN)

The local, most granular level entry name. It has no other qualifying entry names that would serve to uniquely address the entry. In the example, `cn=Smith,o=acme,c=US`, the RDN is `cn=Smith`.

remote master site (RMS)

In a replicated environment, any site, other than the [master definition site \(MDS\)](#), that participates in Oracle9i 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.

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.

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.

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).

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

A

abandoning an operation, 3-42
access control, 2-6, 2-8
 and authorization, 2-8
access control information (ACI), 2-8
 attributes, 2-8
 directives
 format, 2-8
Access Control List (ACL), 2-8
access control lists (ACLs), 2-8
ACI. See access control information (ACI)
ACLs. See Access Control List (ACL)
add.log, 10-7
administration tools
 ldapaddmt, 10-7
 ldapbind, 10-9
 ldapcompare, 10-19
 ldapdelete, 10-10
 ldapmoddn, 10-11
 ldapmodify, 10-22
 ldapmodifymt, 10-27
anonymous authentication, 2-7
applications, building
 with PL/SQL LDAP API, 4-2
 with the C API, 3-63
attribute options
 searching for by using ldapsearch, 10-16
attributes
 adding
 concurrently, by using ldapaddmt, 10-7
 to existing entries, 10-4
 attribute options
 searching for by using ldapsearch, 10-16

 deleting
 by using ldapmodify, 10-25
 values, by using ldapmodify, 10-25
 in LDIF files, 10-2
 types, 2-5
 values, 2-5
 replacing, by using ldapmodify, 10-25
authentication, 2-6
 anonymous, 2-7
 certificate-based, 2-7
 Kerberos, 10-5, 10-8, 10-10
 modes, SSL, 3-2
 one-way SSL, 2-7
 options, 2-7
 password-based, 2-7
 PKI, 2-9
 SSL, 2-7, 3-2, 10-6, 10-8, 10-9, 10-23, 10-28
 none, 3-2
 one-way, 3-2
 two-way, 3-2
 strong, 2-7
 to a directory server
 enabling, 2-16
 enabling, by using DBMS_LDAP, 2-17
 enabling, by using the C API, 2-16
 to the directory, 3-17
 two-way SSL, 2-7
authorization, 2-6, 2-8
authorization ID, 2-6

B

bulk tools, 1-2

C

C API, 3-1

functions

- abandon, 3-42
 - abandon_ext, 3-42
 - add, 3-36
 - add_ext, 3-36
 - add_ext_s, 3-36
 - add_s, 3-36
 - compare, 3-26
 - compare_ext, 3-26
 - compare_ext_s, 3-26
 - compare_s, 3-26
 - count_entries, 3-51
 - count_references, 3-51
 - count_values, 3-55
 - count_values_len, 3-55
 - delete, 3-38
 - delete_ext, 3-38
 - delete_ext_s, 3-38
 - delete_s, 3-38
 - dn2ufn, 3-57
 - err2string, 3-46
 - explode_dn, 3-57
 - explode_rdn, 3-57
 - extended_operation, 3-40
 - extended_operation_s, 3-40
 - first_attribute, 3-53
 - first_entry, 3-51
 - first_message, 3-49
 - first_reference, 3-51
 - get_dn, 3-57
 - get_entry_controls, 3-59
 - get_option, 3-10
 - get_values, 3-55
 - get_values_len, 3-55
 - init, 3-9
 - init_ssl call, 3-3
 - modify, 3-30
 - modify_ext, 3-30
 - modify_ext_s, 3-30
 - modify_s, 3-30
 - msgfree, 3-43
 - msgid, 3-43
 - msgtype, 3-43
 - next_attribute, 3-53
 - next_entry, 3-51
 - next_message, 3-49
 - next_reference, 3-51
 - open, 3-9
 - parse_extended_result, 3-46
 - parse_reference, 3-60
 - parse_result, 3-46
 - parse_sasl_bind_result, 3-46
 - rename, 3-33
 - rename_s, 3-33
 - result, 3-43
 - sasl_bind, 3-17
 - sasl_bind_s, 3-17
 - search, 3-21
 - search_ext, 3-21
 - search_ext_s, 3-21
 - search_s, 3-21
 - search_st, 3-21
 - set_option, 3-10
 - simple_bind, 3-17
 - simple_bind_s, 3-17
 - unbind, 3-20
 - unbind_ext, 3-20
 - unbind_s, 3-20
 - value_free, 3-55
 - value_free_len, 3-55
 - reference, 3-4
 - sample search tool, 3-63
 - sample usage, 3-61
 - summary, 3-4
 - usage with SSL, 3-61
 - usage without SSL, 3-62
- Catalog Management Tool
syntax, 10-18
- Catalog Management tool
syntax, 10-18
- catalog.sh
syntax, 10-18
- catldap.sql, 4-2
- certificate authority, 2-7
- certificate-based authentication, 2-7
- certificates, 2-7
- change types, in ldapmodify input files, 10-24

- changetype
 - add, 10-24
 - delete, 10-25
 - modify, 10-24
 - modrdn, 10-25
- children of an entry, listing, 3-26
- command line tools
 - ldapaddmt, 10-7
 - ldapbind, 10-9
 - ldapcompare, 10-19
 - ldapdelete, 10-10
 - ldapmoddn, 10-11
 - ldapmodify, 10-22
 - ldapmodifymt, 10-27
 - ldapsearch, 10-13
- command-line tools
 - syntax, 10-4
- components
 - Oracle Internet Directory SDK, 1-2
- controls, working with, 3-15

D

- data
 - integrity, 2-6, 2-8
 - privacy, 2-6, 2-9
- data-type summary, 4-9
- DBMS_LDAP
 - about, 4-1
 - building applications with, 4-2
 - sample usage
 - about, A-1
 - for a search, A-10
 - from a database trigger, A-2
 - Java sample code, A-34
- DBMS_LDAP package, 2-11, 4-1
 - searching by using, 2-18
- DBMS_LDAP_UTL
 - about, 7-1
 - data-types, 7-47
 - function return codes, 7-45
 - group-related subprograms
 - about, 7-3
 - function create_group_handle, 7-23
 - function get_group_dn, 7-28

- function get_group_properties, 7-26
- function set_group_handle_properties, 7-24
- miscellaneous subprograms
 - about, 7-4
 - function check_interface_version, 7-44
 - function create_mod_propertyset, 7-41
 - function get_property_names, 7-36
 - function get_property_values, 7-38
 - function get_property_values_len, 7-39
 - function normalize_dn_with_case, 7-35
 - function populate_mod_propertyset, 7-42
 - procedure free_handle, 7-44
 - procedure free_mod_propertyset, 7-43
 - procedure free_propertyset_collection, 7-40
- reference, 7-2
- subscriber-related subprograms
 - about, 7-4
 - function create_subscriber_handle, 7-30
 - function get_subscriber_dn, 7-33
 - function get_subscriber_properties, 7-31
- user-related subprograms
 - about, 7-3
 - function authenticate_user, 7-6
 - function check_group_membership, 7-18
 - function create_user_handle, 7-8
 - function get_group_membership, 7-21
 - function get_user_dn, 7-16
 - function get_user_extended_properties, 7-15
 - function get_user_properties, 7-11
 - function locate_subscriber_for_user, 7-19
 - function set_user_handle_properties, 7-9
 - function set_user_properties, 7-13
- deleting values from attributes, 10-25
- dependencies and limitations, 3-76, 4-2
 - C API, 3-76
 - PL/SQL API, 4-2
- DES40 encryption, 2-9
- directives, 2-8
- directory information tree (DIT), 2-2
- distinguished names, 2-2
 - components of, 2-3
 - format, 2-3
 - in LDIF files, 10-2
- DNs. see distinguished names.
- documentation, related, xvii

E

encryption

- DES40, 2-9
- levels available in Oracle Internet Directory, 2-9
- options for passwords, 2-9
- passwords, 2-9
 - default, 2-9
 - MD4, 2-9
 - MD5, 2-9
 - SHA, 2-9
 - UNIX crypt, 2-9
- RC4_40, 2-9

entries

- adding
 - by using ldapaddmt, 10-7
 - concurrently, 10-7
- deleting
 - by using ldapdelete, 10-10
 - by using ldapmodify, 10-25
- distinguished names of, 2-2
- locating by using distinguished names, 2-3
- modifying
 - by using ldapmodify, 10-22
 - concurrently by using ldapmodifymt, 10-27
- naming, 2-2
- reading, 3-26

errors

- handling and parsing results, 3-46

examples of ldapsearch filters, 10-15

exception summary, 4-6

F

filters, 2-23

- IETF-compliant, 10-13
- ldapsearch, 10-15

formats, of distinguished names, 2-3

G

group entries, creating by using ldapmodify, 10-24

H

header files and libraries, required, 3-63

history of LDAP, 2-2

I

integrity, data, 2-8

interface calls, SSL, 3-3

J

Java, 1-2

Java API reference

- about, 6-1
- class descriptions, 6-2
 - group class, 6-4
 - Property class, 6-5
 - PropertySet class, 6-5
 - PropertySetCollection class, 6-5
 - subscriber class, 6-3
 - user class, 6-2
- classes, 6-6
- exceptions, 6-69

JNDI, 1-2

jpeg images, adding with ldapadd, 10-7

K

Kerberos authentication, 10-5, 10-8, 10-10

L

LDAP

- data interchange format (LDIF), 10-2
 - syntax, 10-2
- functional model, 2-5
- history, 2-2
- information model, 2-4
- messages, obtaining results and peeking
 - inside, 3-43
- naming model, 2-2
- operations, performing, 3-21
- search filters, IETF-compliant, 10-13
- security model, 2-6
- session handle options, 3-10
 - in the C API, 2-16
- sessions
 - initializing, 2-14, 3-9

- version 2 C API, 3-2
- ldapadd
 - adding jpeg images, 10-7
- ldapaddmt, 10-7
 - adding entries concurrently, 10-7
 - log, 10-7
 - syntax, 10-7
- ldapbind, 10-9
 - syntax, 10-9
- ldap-bind operation, 2-6
- ldapcompare, 10-19
 - syntax, 10-19, 10-20
- ldapdelete, 10-10
 - deleting entries, 10-10
 - syntax, 10-10
- ldapmoddn, 10-11
 - syntax, 10-11
- ldapmodify, 10-22
 - adding values to multivalued attributes, 10-24
 - change types, 10-24
 - creating group entries, 10-24
 - deleting entries, 10-25
 - LDIF files in, 10-4, 10-7, 10-22, 10-27
 - replacing attribute values, 10-25
 - syntax, 10-22
- ldapmodifymt, 10-27
 - by using, 10-27
 - multithreaded processing, 10-28
 - syntax, 10-27
- ldapsearch, 3-63
 - filters, 10-15
 - syntax, 10-13
- LDIF
 - by using, 10-2
 - files, in ldapmodify commands, 10-4, 10-7, 10-22, 10-27
 - formatting notes, 10-3
 - formatting rules, 10-3
 - syntax, 10-2

M

- MD4, for password encryption, 2-9
- MD5, for password encryption, 2-9
- multiple threads, 10-28

- in ldapaddmt, 10-7
- increasing the number of, 10-7
- multithreaded command line tools
 - ldapaddmt, 10-7
 - ldapmodifymt, 10-28
- multivalued attributes, adding values to, 10-24

N

- naming entries, 2-2

O

- object classes
 - adding concurrently by using ldapaddmt, 10-7
 - in LDIF files, 10-2
- objects, removing, 10-10, 10-22
- one-way SSL authentication, 2-7, 3-2
- OpenLDAP Community, xviii
- operating systems supported by Oracle Internet Directory, 1-3
- operational attributes
 - ACI, 2-8
- Oracle Directory Manager, 1-2
 - listing attribute types, 10-3
- Oracle directory replication server, 1-2
- Oracle directory server, 1-2
- Oracle Extensions
 - LDAP access model, 5-2
- Oracle extensions
 - about, 5-1
 - API enhancements
 - assumptions, 5-6
 - functional categorization, 5-7
 - overview and usage model, 5-6
 - usage model, 5-8
 - application
 - deinstallation logic, 5-4
 - installation logic, 5-3
 - runtime logic, 5-3
 - shutdown logic, 5-4
 - startup and bootstrap logic, 5-3
- entities modeled in LDAP
 - about, 5-4
 - groups, 5-5

- subscribers, 5-5
 - users, 5-5
- programming abstractions
 - for Java language, 5-10
 - for PL/SQL language, 5-9
- user management functionality, 5-10, 5-11
- Oracle extensions to support SSL, 3-2
- Oracle instances, Glossary-22
- Oracle Internet Directory, components, 1-2
- Oracle SSL call interface, 3-2, 4-2
- Oracle SSL extensions, 3-2
- Oracle SSL-related libraries, 3-77
- Oracle system libraries, 3-77
- Oracle wallet, 3-3
- Oracle Wallet Manager, 3-3
 - required for creating wallets, 3-76
- Oracle wallet parameter
 - modifying, 10-6, 10-8, 10-9, 10-11, 10-13, 10-15, 10-21, 10-23, 10-28
- Oracle wallets, changing location of, 10-6, 10-8, 10-9, 10-11, 10-13, 10-15, 10-21, 10-23, 10-28
- overview of LDAP models, 2-2

P

- password-based authentication, 2-7
- passwords
 - encryption, 2-6, 2-9
 - default, 2-9
 - MD4, 2-9
 - MD5, 2-9
 - SHA, 2-9
 - UNIX crypt, 2-9
 - encryption options, 2-9
 - policies, 2-10
- performance
 - increasing, by using multiple threads, 10-7
- permissions, 2-6, 2-8
- PKI authentication, 2-9
- PL/SQL API, 4-1, 4-2
 - contains subset of C API, 2-11
 - data-type summary, 4-9
 - dependencies and limitations, 4-2
 - exception summary, 4-6
 - functions

- add_s, 4-55
- ber_free, 4-68
- bind_s, 4-14
- compare_s, 4-18
- count_entries, 4-30
- count_values, 4-58
- count_values_len, 4-59
- create_mod_array, 4-47
- dbms_ldap.init, 4-11
- delete_s, 4-42
- err2string, 4-46
- explode_dn, 4-62
- first_attribute, 4-32
- first_entry, 4-26
- get_dn, 4-36
- get_values, 4-38
- get_values_len, 4-40
- init, 4-10
- modify_s, 4-53
- modrdn2_s, 4-44
- msgfree, 4-66
- next_attribute, 4-34
- next_entry, 4-28
- open_ssl, 4-64, 4-66, 4-68
- rename_s, 4-60
- search_s, 4-20
- search_st, 4-23
- simple_bind_s, 4-12
- unbind_s, 4-16
- loading into database, 4-2
- procedures
 - free_mod_array, 4-57
 - populate_mod_array (binary version), 4-51
 - populate_mod_array (string version), 4-49
- reference, 4-3
- subprograms, 4-10
- summary, 4-3
- using for a search, A-10
- using from a database trigger, A-2
- privacy, data, 2-6, 2-9
- privileges, 2-6, 2-8
- procedures, PL/SQL
 - free_mod_array, 4-57
 - populate_mod_array (binary version), 4-51
 - populate_mod_array (string version), 4-49

- provisioning tool
 - syntax, 10-29
- public key
 - infrastructure, 2-9

R

- RC4_40 encryption, 2-9
- RDNs. see relative distinguished names (RDNs)
- related documentation, xvii
- relative distinguished names (RDNs), 2-3
 - modifying by using ldapmodify, 10-25
- results, stepping through a list of, 3-49
- RFC 1823, 3-77
- rules, LDIF, 10-3

S

- sample C API usage, 3-61
- sample PL/SQL usage, 4-2
- sample search tool, building with C API, 3-63
- SDK components, 1-2
- search
 - filters
 - IETF-compliant, 10-13
 - ldapsearch, 10-15
 - results
 - parsing, 3-50
 - scope, 2-22
 - search-related operations, flow of, 2-19
- security, within Oracle Internet Directory environment, 2-6
- sessions
 - closing, 3-20
 - enabling termination by using DBMS_LDAP, 2-24
 - initializing
 - by using DBMS_LDAP, 2-15
 - by using the C API, 2-14
- session-specific user identity, 2-6
- SHA (Secure Hash Algorithm), for password encryption, 2-9
- simple authentication, 2-7
- Smith, Mark, xviii
- SQL*Plus, 4-2

SSL

- authentication modes, 3-2
- default port, 2-7
- enabling, 10-6, 10-8, 10-9, 10-23, 10-28
- handshake, 3-3
- interface calls, 3-3
- modifying orclsslwalleturl parameter, 10-6, 10-8, 10-9, 10-11, 10-13, 10-15, 10-21, 10-23, 10-28
- no authentication, 2-7
- one-way authentication, 2-7
- Oracle extensions, 3-2
 - provide encryption and decryption, 3-2
- strong authentication, 2-9
- two-way authentication, 2-7
- wallets, 3-3
 - changing location of, 10-6, 10-8, 10-9, 10-11, 10-13, 10-15, 10-21, 10-23, 10-28

strong authentication, 2-7

syntax

- Catalog Management Tool, 10-18
- catalog management tool, 10-18
- catalog.sh, 10-18
- command-line tools, 10-4
- ldapaddmt, 10-7
- ldapbind, 10-9
- ldapcompare, 10-19, 10-20
- ldapdelete, 10-10
- ldapmoddn, 10-11
- ldapmodify, 10-22
- ldapmodifymt, 10-27
- ldapsearch, 10-13
- LDIF, 10-2
- LDIF and command-line tools, A-1
- provisioning tool, 10-29

T

- TCP/IP socket library, 3-76
- two-way authentication, SSL, 3-2
- types of attributes, 2-5

U

- UNIX crypt, for password encryption, 2-9

W

wallets

changing location of, 10-6, 10-8, 10-9, 10-11,
10-13, 10-15, 10-21, 10-23, 10-28

SSL, 3-3

support, 3-3