Oracle® Internet Directory

Application Developer's Guide

Release 3.0.1

June 2001

Part No. A90152-01

ORACLE®

Oracle Internet Directory Application Developer's Guide, Release 3.0.1

Part No. A90152-01

Primary Author: Richard Smith

Contributing Authors: Henry Abrecht, Ginger Tabora

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

Graphic Designer: Valarie Moore

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

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.

Contents

3 The Oracle Internet Directory C API

Send Us Your Comments

Oracle Internet Directory Application Developer's Guide, Release 3.0.1

Part No. A90152-01

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

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 and PL/SQL API work.

Organization

Chapter 1, "Introduction"

Briefly describes the intended audience and components of Oracle Internet Directory Software Developer's Kit release 3.0.1. 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, "The Oracle Internet Directory C API"

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

Chapter 4, "The Oracle Internet Directory PL/SQL API"

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, "Command-Line Tools Syntax"

Provides syntax, usage notes, and examples for using LDAP Data Interchange Format (LDIF) and LDAP command line tools

Glossary

Related Documentation

For more information, see these Oracle resources:

- Oracle9*i* 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://technet.oracle.com/membership/index.htm`

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://technet.oracle.com/docs/index.htm`

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) documenation, 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**.
Italics	Italic typeface indicates book titles or emphasis.	*Oracle9i Database Concepts* Ensure that the recovery catalog and target database do *not* reside on the same disk.

Convention	Meaning	Example
UPPERCASE monospace (fixed-width font)	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, usernames, and roles.	You can specify this clause only for a NUMBER column. You can back up the database by using the BACKUP command. Query the TABLE_NAME column in the USER_TABLES data dictionary view. Use the DBMS_STATS.GENERATE_STATS procedure.
lowercase monospace (fixed-width font)	Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter sqlplus to open SQL*Plus. The password is specified in the orapwd file. Back up the datafiles and control files in the /disk1/oracle/dbs directory. The department_id, department_name, and location_id columns are in the hr.departments table. Set the QUERY_REWRITE_ENABLED initialization parameter to true. Connect as oe user. The JRepUtil class implements these methods.
lowercase monospace (fixed-width font) italic	Lowercase monospace italic font represents placeholders or variables.	You can specify the *parallel_clause*. Run U*old_release*.SQL where *old_release* refers to the release you installed prior to upgrading.

Conventions in Code Examples

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

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

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

Convention	Meaning	Example
[ ]	Brackets enclose one or more optional items. Do not enter the brackets.	`DECIMAL (digits [ , precision ])`
{ }	Braces enclose two or more items, one of which is required. Do not enter the braces.	`{ENABLE | DISABLE}`
|	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	`{ENABLE | DISABLE}` `[COMPRESS | NOCOMPRESS]`
...	Horizontal ellipsis points indicate either:	
	■ That we have omitted parts of the code that are not directly related to the example	`CREATE TABLE ... AS subquery;`
	■ That you can repeat a portion of the code	`SELECT col1, col2, ... , coln FROM employees;`
. . .	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.	`acctbal NUMBER(11,2);` `acct CONSTANT NUMBER(4) := 3;`
Italics	Italicized text indicates placeholders or variables for which you must supply particular values.	`CONNECT SYSTEM/system_password` `DB_NAME = database_name`

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

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

Documentation Accessibility

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

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

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.

1

Introduction

This chapter briefly describes the intended audience and components of Oracle Internet Directory Software Developer's Kit release 3.0.1. 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 3.0.1

Oracle Internet Directory SDK release 3.0.1 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 3.0.1 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 3.0.1, 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 95
  - Windows 98
  - Windows 2000
- HPUX
- AIX
- Compaq TRU64
- Intel Solaris
- SGI
- DGUX
- UNIXWARE

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

This chapter contains these topics:

History of LDAP

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

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

Overview of LDAP Models

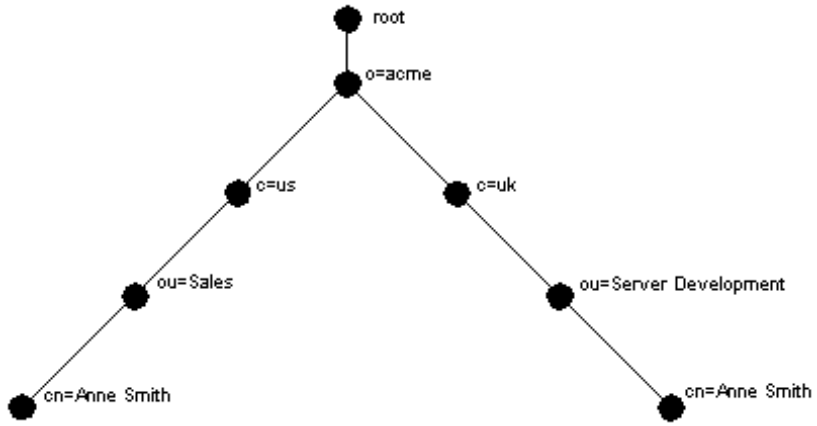
LDAP defines four basic models to describe its operations. This section contains these topics:

LDAP Naming Model

The LDAP naming model allows directory information to be referenced and organized. Each entry in a directory is uniquely identified by a **distinguished name (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

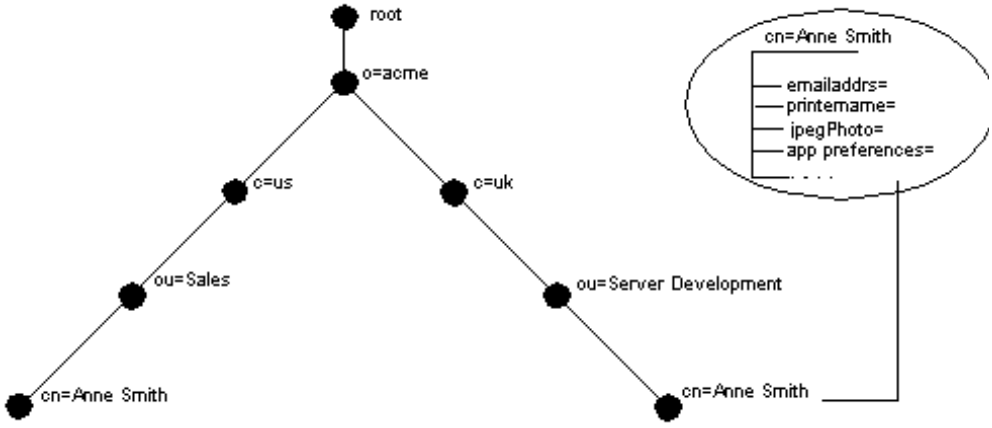
The LDAP information model determines the form and character of information in the directory. It is centered around entries, which are composed of attributes. In a directory, each collection of information about an object is called an **entry**. For example, a typical telephone directory includes entries 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, these information items are called **attributes**. Attributes in a typical employee entry can include, for example, a job title, an e-mail address, or a phone number.

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

`jpegPhoto`, and `app preferences`. Moreover, each bullet in Figure 2–2 is also an entry with attributes, although the attributes for each are not shown.

Figure 2–2 Attributes of the Entry for Anne Smith



Each attribute consists of an attribute type and one or more attribute values. The **attribute type** is the kind of information that the attribute contains—for example, `jobTitle`. The **attribute value** is the particular occurrence of information appearing in that entry. For example, the value for the `jobTitle` attribute could be `manager`.

LDAP Functional Model

The LDAP functional model determines what operations can be performed on the information. There are three types of functions:

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 Sockets 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 points from which such access control policies apply are called **access control policy points** (**ACPs**).

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

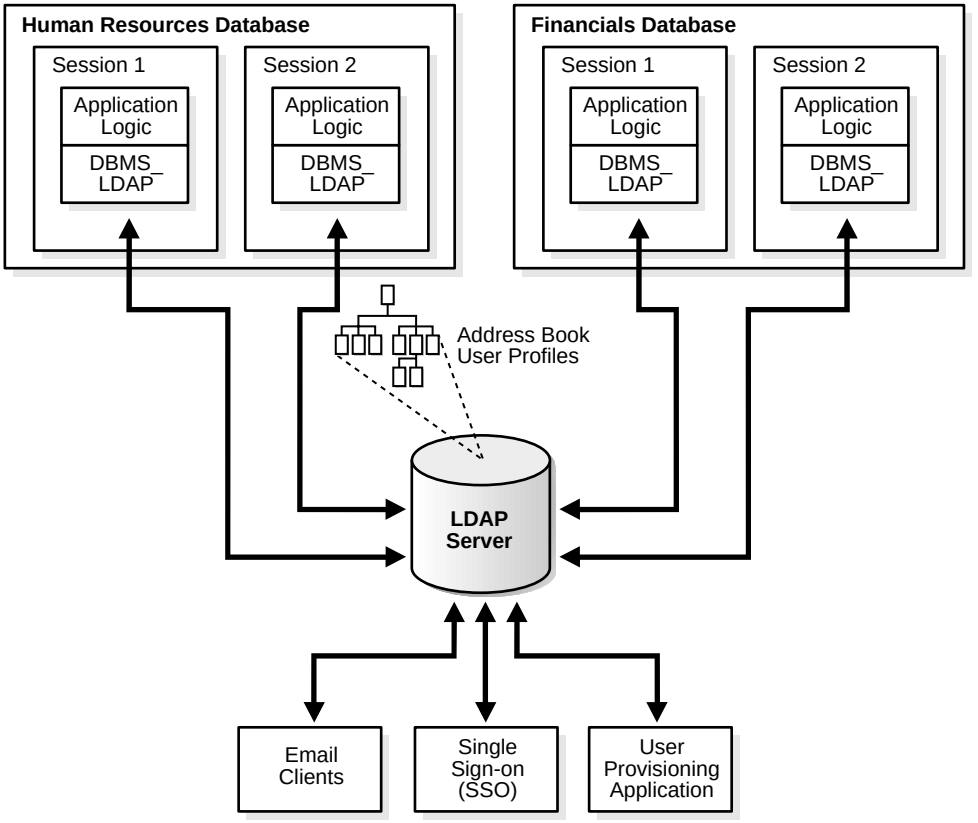
About the Oracle Internet Directory API

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:

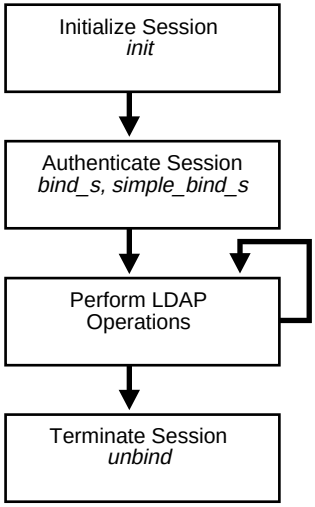
- Read from the LDAP server information that is published by other programs in the enterprise
- Publish in the LDAP server new information that can be used later by the same application or other applications
- Modify or update existing information in the LDAP server based on certain pre-defined conditions

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

Parameter	Description
hostname	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. 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.
portno	Contains the TCP port number to connect to. The default LDAP port of 389 can be obtained by supplying the constant LDAP_PORT. If a host includes a port number then this parameter is ignored.

ldap_init() and ldap_open() both return a session handle, that is, a pointer to an opaque structure that MUST be passed to subsequent calls pertaining to the session. These routines return NULL if the session cannot be initialized in which case the operating system error reporting mechanism can be checked to see why the call failed.

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 per formed 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 [12]. 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:** The appendix about Oracle Wallet Manager in *Oracle Internet Directory Administrator's Guide*

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

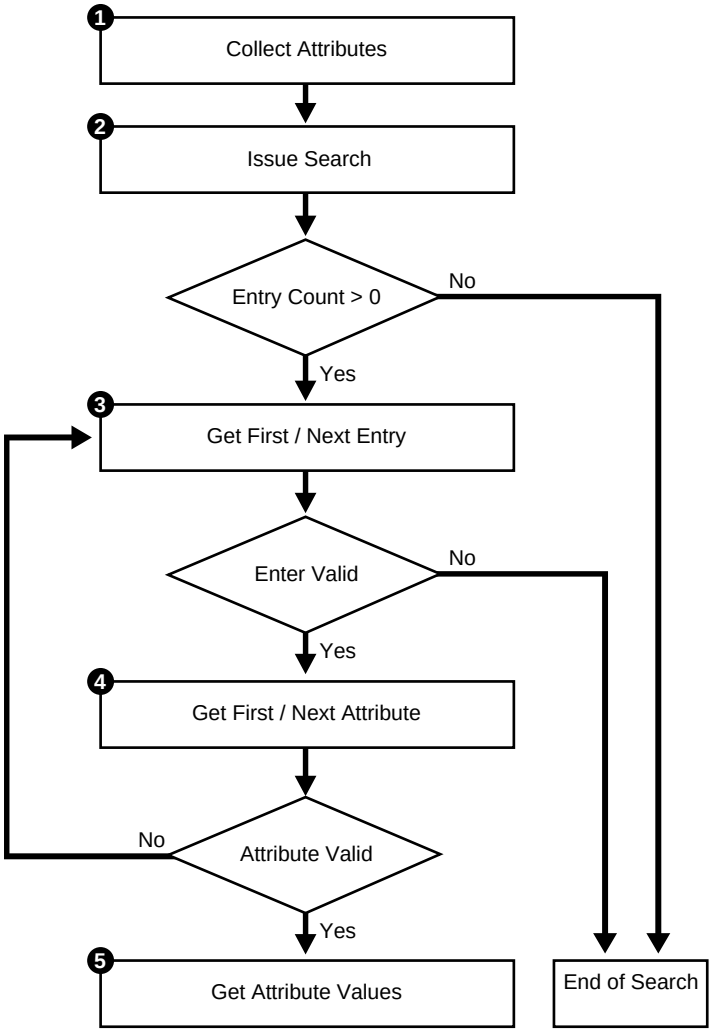
The programming work required to initiate a typical search operation and retrieve results can be broken down into the following steps:

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



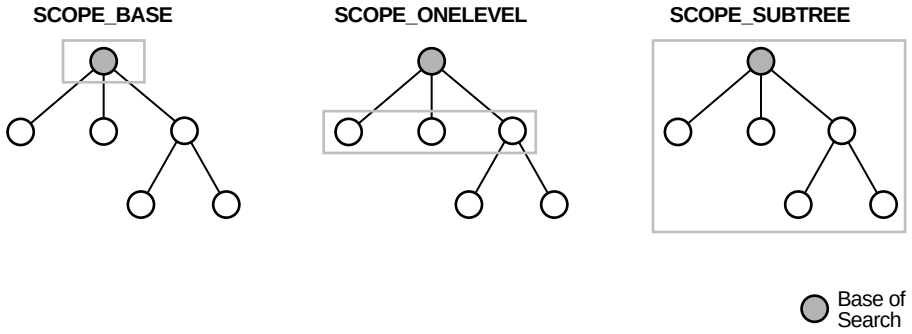
Search Scope

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

The basic filters in Table 2–2 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`.

3

The Oracle Internet Directory C API

This chapter introduces the Oracle Internet Directory C API and provides examples of how to use it. It contains these topics:

About the Oracle Internet Directory C API

The Oracle Internet Directory SDK C API is based on:

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

You can use the Oracle Internet Directory API release 3.0.1 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.

> **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

Oracle SSL extensions to the LDAP API are based on standard SSL protocol. The SSL extensions provide encryption and decryption of data over the wire, and authentication.

There are three modes of authentication:

- None—Neither client nor server is authenticated, and only SSL encryption is used
- One-way—Only the server is authenticated by the client
- Two-way—Both the server and the client are authenticated by each other

The type of authentication is indicated by a parameter in the SSL interface call.

SSL Interface Calls

There is only one call required to enable SSL:

```
int ldap_init_SSL(Sockbuf *sb, text *sslwallet, text *sslwalletpasswd, int
sslauthmode)
```

The ldap_init_SSL call performs the necessary handshake between client and server using the standard SSL protocol. If the call is successful, all subsequent communication happens over a secure connection.

Argument	Description
sb	Socket buffer handle returned by the *ldap_open* call as part of LDAP handle.
sslwallet	Location of the user wallet.
sslwalletpasswd	Password required to use the wallet.
sslauthmode	SSL authentication mode user wants to use. Possible values are: ■ GSLC_SSL_NO_AUTH—No authentication required ■ GSLC_SSL_ONEWAY_AUTH—Only server authentication required. ■ GSLC_SSL_TWOWAY_AUTH—Both server and client authentication required. A return value of 0 indicates success. A non zero return value indicates an error. The error code can be decoded by using the function ldap_err2string.

See Also: See "Sample C API Usage" on page 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 3.0.1 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

Table 3–1 DBMS_LDAP API Subprograms

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

Table 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 ldap_open	Open a connection to an LDAP server
ldap_memfree()	Free memory allocated by an LDAP API function call
ldap_modify_ext ldap_modify_ext_s ldap_modify ldap_modify_s	Modify an entry in the directory
ldap_msgfree	Free the memory allocated for search results or other LDAP operation results
ldap_next_attribute	Get the name of the next attribute in an entry
ldap_next_entry	Get the next entry in a chain of search results
ldap_perror DEPRECATED	Prints the message supplied in message.
ldap_rename ldap_rename_s	Modify the RDN of an entry in the directory
ldap_result2error DEPRECATED	Returns the error code from result message.
ldap_result ldap_msgfree ldap_msgtype ldap_msgid	Check the results of an asynchronous operation
ldap_sasl_bind ldap_sasl_bind_s	General authentication to an LDAP server

Table 3–1 DBMS_LDAP API Subprograms

Function or Procedure	Description
ldap_search_ext	Search the directory
ldap_search_ext_s	
ldap_search	
ldap_search_s	
ldap_search_st	Search the directory with a timeout value
ldap_set_option	Set the value of these parameters
ldap_simple_bind	Simple authentication to an LDAP server
ldap_simple_bind_s	
ldap_unbind_ext	End an LDAP session
ldap_unbind	
ldap_unbind_s	
ldap_value_free	Free the memory allocated for the string values of an attribute
ldap_value_free_len	Free the memory allocated for the binary values of an attribute

This section lists all the calls available in the LDAP C API found in RFC 1823.

See Also: The following URL: 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_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	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. 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.
portno	Contains the TCP port number to connect to. The default LDAP port of 389 can be obtained by supplying the constant LDAP_PORT. If a host includes a port number then this parameter is ignored.

Usage Notes

ldap_init() and ldap_open() both return a "session handle," a pointer to an opaque structure that MUST be passed to subsequent calls pertaining to the session. These routines return NULL if the session cannot be initialized in which case the

operating system error reporting mechanism can be checked to see why the call failed.

Note that if you connect to an LDAPv2 server, one of the LDAP bind calls described below SHOULD be completed before other operations can be per formed 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 below.

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 ldap_init() or ldap_open() inherit their characteristics from these global defaults.
option	The name of the option being accessed or set. This parameter SHOULD be one of the constants listed and described in Table 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 char ** and LDAPControl **, a copy of the data that is associated with the LDAP session ld is returned; callers should dispose of the memory by calling ldap_memfree() or ldap_controls_free(), depending on the type of data returned.

Table 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 non-zero 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 abort 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 non-zero 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 the Working With Controls section below.
LDAP_OPT_ CLIENT_ CONTROLS (0x13)	LDAPControl **	LDAPControl ***	A default list of client controls that affect the LDAP session. See the Working With Controls section below.
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 below 0x1000 and above 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
ldctl_oid	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 of not. If this field is non-zero, 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 (see above). 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 basic by using the ldap_set_option() function with the LDAP_OPT_REFERRALS option. It is also useful to govern automatic referral chasing on per-request basis. A client control with an OID of 1.2.840.113556.1.4.616 exists to provide this functionality.

```
/* OID for referrals client control */
#define LDAP_CONTROL_REFERRALS              "1.2.840.113556.1.4.616"

/* Flags for referrals client control value */
#define LDAP_CHASE_SUBORDINATE_REFERRALS    0x00000020U
#define LDAP_CHASE_EXTERNAL_REFERRALS       0x00000040U
```

To create a referrals client control, the ldctl_oid field of an LDAPControl structure MUST be set to LDAP_CONTROL_REFERRALS ("1.2.840.113556.1.4.616") and the ldctl_value field MUST be set to a 4-octet value that contains a set of flags. The ldctl_value.bv_len field MUST always be set to 4. The ldctl_value.bv_val field MUST point to a 4-octet integer flags value. This flags value can be set to zero to disable automatic chasing of referrals and LDAPv3 references altogether. Alternatively, the flags value can be set to the value LDAP_CHASE_ SUBORDINATE_REFERRALS (0x00000020U) to indicate that only LDAPv3 search continuation references are to be automatically chased by the API implementation, to the value LDAP_CHASE_EXTERNAL_REFERRALS (0x00000040U) to indicate that only LDAPv3 referrals are to be automatically chased, or the logical OR of the two flag values (0x00000060U) to indicate that both referrals and references are to be automatically chased.

Authenticating to the Directory

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

ldap_sasl_bind

ldap_sasl_bind_s

ldap_simple_bind

ldap_simple_bind_s

The ldap_sasl_bind() and ldap_sasl_bind_s() functions can be used to do general and extensible authentication over LDAP through the use of the Simple Authentication Security Layer. The routines both take the dn to bind as, the method to use, as a dotted-string representation of an object identifier identifying the method, and a struct berval holding the credentials. The special constant value

LDAP_SASL_SIMPLE (NULL) can be passed to request simple authentication, or the simplified routines ldap_simple_bind() or ldap_simple_bind_s() can be used.

Syntax

```
int ldap_sasl_bind
(
    LDAP                    *ld,
    const char              *dn,
    const char              *mechanism,
    const struct berval     *cred,
    LDAPControl             **serverctrls,
    LDAPControl             **clientctrls,
    int                     *msgidp
);

int ldap_sasl_bind_s(
    LDAP                    *ld,
    const char              *dn,
    const char              *mechanism,
    const struct berval     *cred,
    LDAPControl             **serverctrls,
    LDAPControl             **clientctrls,
    struct berval           **servercredp
);


int ldap_simple_bind(
    LDAP                    *ld,
    const char              *dn,
    const char              *passwd
);


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

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

```
int ldap_bind( LDAP *ld, const char *dn, const char *cred, int method );

int ldap_bind_s( LDAP *ld, const char *dn, const char *cred, int method );

int ldap_kerberos_bind( LDAP *ld, const char *dn );

int ldap_kerberos_bind_s( LDAP *ld, const char *dn );
```

Parameters

Table 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. See the section below on error handling for more information

about possible errors and how to interpret them. If successful, ldap_sasl_bind() places the message id of the request in *msgidp. A subsequent call to ldap_result(), described below, 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 below, 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. See the section below on error handling for more information about possible errors and how to interpret them.

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().

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 non-zero 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 non-zero, 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	This result parameter will be set to the message id of the request if the ldap_search_ext() call succeeds.There are three options in the session handle ld which potentially affect how the search is performed. They are: ■ LDAP_OPT_SIZELIMIT—A limit on the number of entries to return from the search. A value of LDAP_NO_LIMIT (0) means no limit. Note that the value from the session handle is ignored when using the ldap_search_ext() or ldap_search_ext_s() functions. ■ LDAP_OPT_TIMELIMIT—A limit on the number of seconds to spend on the search. A value of LDAP_NO_ LIMIT (0) means no limit. Note that the value from the session handle is ignored when using the ldap_search_ ext() or ldap_search_ext_s() functions. ■ LDAP_OPT_DEREF—One of LDAP_DEREF_NEVER (0x00), LDAP_DEREF_SEARCHING (0x01), LDAP_ DEREF_FINDING (0x02), or LDAP_DEREF_ALWAYS (0x03), specifying how aliases are handled during the search. The LDAP_DEREF_SEARCHING value means aliases are dereferenced during the search but not when locating the base object of the search. The LDAP_DEREF_ FINDING value means aliases are dereferenced when locating the base object but not during the search.

Usage Notes

The ldap_search_ext() function initiates an asynchronous search operation and returns the constant LDAP_SUCCESS if the request was successfully sent, or another LDAP error code if not. See the section below on error handling for more information about possible errors and how to interpret them. If successful, ldap_ search_ext() places the message id of the request in *msgidp. A subsequent call to ldap_result(), described below, 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 below, 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. See the section below on error handling for more information about possible errors and how to interpret them. 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. 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 abort the search operation.

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. See the section below on error handling for more information about possible errors and how to interpret them. If successful, ldap_compare_ext() places the message id of the request in *msgidp. A subsequent call to ldap_result(), described below, 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

below, 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. See the section below on error handling for more information about possible errors and how to interpret them.

The ldap_compare_ext() and ldap_compare_ext_s() functions support LDAPv3 server controls and client controls.

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

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

Table 3–11

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

Usage Notes

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

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

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

The ldap_modify_ext() function initiates an asynchronous modify operation and returns the constant LDAP_SUCCESS if the request was successfully sent, or another LDAP error code if not. See the section below on error handling for more information about possible errors and how to interpret them. If successful, ldap_modify_ext() places the message id of the request in *msgidp. A subsequent call to ldap_result(), described below, 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 below, 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. See the section below on error handling for more information about possible errors and how to interpret them.

The ldap_modify_ext() and ldap_modify_ext_s() functions support LDAPv3 server controls and client controls.

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 non-zero 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. See the section below on error handling for more information about possible errors and how to interpret them. If successful, ldap_rename() places the DN message id of the request in *msgidp. A subsequent call to ldap_result(), described below, 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. See the section below on error handling for more information about possible errors and how to interpret them.

The ldap_rename() and ldap_rename_s() functions both support LDAPv3 server controls and client controls.

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 ldap_modify(). The mod_type and mod_vals fields MUST be filled in. The mod_op field is ignored unless ORed with the constant LDAP_MOD_BVALUES, used to select the mod_bvalues case of the mod_vals union.
serverctrls	List of LDAP server controls.
clientctrls	List of client controls.
msgidp	This result parameter will be set to the message id of the request if the ldap_add_ext() call succeeds.

Usage Notes

Note that the parent of the entry being added must already exist or the parent must be empty (i.e., 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. See the section below on error handling for more information about possible errors and how to interpret them. If successful, ldap_add_ext() places the message id of the request in *msgidp. A subsequent call to ldap_result(), described below, 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 below, 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. See the section below on error handling for more information about possible errors and how to interpret them.

The ldap_add_ext() and ldap_add_ext_s() functions support LDAPv3 server controls and client controls.

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 ldap_delete_ext() call succeeds.

Usage Notes

Note that the entry to delete must be a leaf entry (i.e., 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. See the section below on error handling for more information about possible errors and how to interpret them. If successful, ldap_delete_ext() places the message id of the request in *msgidp. A subsequent call to ldap_result(), described below, 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 below, 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. See the section below on error handling for more information about possible errors and how to interpret them.

The ldap_delete_ext() and ldap_delete_ext_s() functions support LDAPv3 server controls and client controls.

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 ldap_extended_operation() call succeeds.
retoidp	Pointer to a character string that will be set to an allocated, dotted-OID text string returned by the server. This string SHOULD be disposed of using the ldap_memfree() function. If no OID was returned, *retoidp is set to NULL.
retdatap	Pointer to a berval structure pointer that will be set an allocated copy of the data returned by the server. This struct berval SHOULD be disposed of using ber_bvfree(). If no data is returned, *retdatap is set to NULL.

Usage Notes

The ldap_extended_operation() function initiates an asynchronous extended operation and returns the constant LDAP_SUCCESS if the request was successfully sent, or another LDAP error code if not. See the section below on error handling for more information about possible errors and how to interpret them. If successful, ldap_extended_operation() places the message id of the request in *msgidp. A subsequent call to ldap_result(), described below, 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. See the section below on error handling for more information about possible errors and how to interpret them. 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.

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. See the section below on error handling for more information about possible errors and how to interpret them.

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.

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_MODDN (0x6D)

LDAP_RES_COMPARE (0x6F)

LDAP_RES_EXTENDED (0x78) -- new in LDAPv3

ldap_result() returns 0 if the timeout expired and -1 if an error occurs, in which case the error parameters of the LDAP session handle will be set accordingly.

ldap_msgfree() frees each message in the result chain pointed to by res and returns the type of the last message in the chain. If res is NULL, 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 above, or -1 on error.

ldap_msgid() returns the message ID associated with the LDAP message passed as a parameter, or -1 on error.

Handling Errors and Parsing Results

ldap_parse_result

ldap_parse_sasl_bind_result

ldap_parse_extended_result

ldap_err2string

These calls are used to extract information from results and handle errors returned by other LDAP API routines. Note that ldap_parse_sasl_bind_result() and ldap_parse_extended_result() must typically be used in addition to ldap_parse_result() to retrieve all the result information from SASL Bind and Extended Operations respectively.

Syntax

```
int ldap_parse_result
(
    LDAP            *ld,
    LDAPMessage     *res,
    int             *errcodep,
    char            **matcheddnp,
    char            **errmsgp,
    char            ***referralsp,
    LDAPControl     ***serverctrlsp,
    int             freeit
);


int ldap_parse_sasl_bind_result
(
    LDAP            *ld,
    LDAPMessage     *res,
    struct berval   **servercredp,
    int             freeit
);
```

int ldap_parse_extended_result

```
(
    LDAP            *ld,
    LDAPMessage     *res,
    char            **retoidp,
    struct berval   **retdatap,
    int             freeit
);
#define LDAP_NOTICE_OF_DISCONNECTION    "1.3.6.1.4.1.1466.20036"
char *ldap_err2string( int err );
```

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

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

Parameters

Table 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.
matcheddnp	In the case of a return of LDAP_NO_SUCH_OBJECT, this result parameter will be filled in with a DN indicating how much of the name in the request was recognized. NULL SHOULD be passed to ignore this field. The matched DN string SHOULD be freed by calling ldap_memfree() which is described later in this document.

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 non-zero 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 non-zero, the entire chain of messages represented by res is disposed of.
servercredp	For SASL bind results, this result parameter will be filled in with the credentials passed back by the server for mutual authentication, if given. An allocated berval structure is returned that SHOULD be disposed of by calling ber_bvfree(). NULL SHOULD be passed to ignore this field.
retoidp	For extended results, this result parameter will be filled in with the dotted-OID text representation of the name of the extended operation response. This string SHOULD be disposed of by calling ldap_memfree(). NULL SHOULD be passed to ignore this field. The LDAP_NOTICE_OF_DISCONNECTION macro is defined as a convenience for clients that wish to check an OID to see if it matches the one used for the unsolicited Notice of Disconnection (defined in RFC 2251[2] section 4.4.1).
retdatap	For extended results, this result parameter will be filled in with a pointer to a struct berval containing the data in the extended operation response. It SHOULD be disposed of by calling ber_bvfree(). NULL SHOULD be passed to ignore this field.
err	For ldap_err2string(), an LDAP error code, as returned by ldap_parse_result() or another LDAP API call.

Usage Notes

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

The ldap_parse_result(), ldap_parse_sasl_bind_result(), and ldap_parse_extended_result() functions all skip over messages of type LDAP_RES_SEARCH_ENTRY and LDAP_RES_SEARCH_REFERENCE when looking for a result message to parse. They return the constant LDAP_SUCCESS if the result was successfully parsed and another LDAP error code if not. Note that the LDAP error code that indicates the outcome of the operation performed by the server is placed in the errcodep ldap_parse_result() parameter. If a chain of messages that contains more than one result message is passed to these routines they always operate on the first result in the chain.

ldap_err2string() is used to convert a numeric LDAP error code, as returned by ldap_parse_result(), ldap_parse_sasl_bind_result(), ldap_parse_extended_result() or one of the synchronous API operation calls, into an informative zero-terminated character string message describing the error. It returns a pointer to static data.

Stepping Through a List of Results

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 above, 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 ldap_result().
msg	The message returned by a previous call to ldap_first_message() or ldap_next_message().

Usage Notes

ldap_first_message() and ldap_next_message() will return NULL when no more messages exist in the result set to be returned. NULL is also returned if an error occurs while stepping through the entries, in which case the error parameters in the session handle ld will be set to indicate the error.

If successful, ldap_count_messages() returns the number of messages contained in a chain of results; if an error occurs such as the res parameter being invalid, -1 is returned. The ldap_count_messages() call can also be used to count the number of messages that remain in a chain if called with a message, entry, or reference returned by ldap_first_message(), ldap_next_message(), ldap_first_entry(), ldap_next_entry(), ldap_first_reference(), ldap_next_reference().

Parsing Search Results

The following calls are used to parse the entries and references returned by ldap_search() and friends. These results are returned in an opaque structure that MAY be accessed by calling the routines described below. 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
ld	The session handle.
res	The search result, as obtained by a call to one of the synchronous search routines or ldap_result().
entry	The entry returned by a previous call to ldap_first_entry() or ldap_next_entry().
ref	The reference returned by a previous call to ldap_first_reference() or ldap_next_reference().

Usage Notes

ldap_first_entry(), ldap_next_entry(), ldap_first_reference() and ldap_next_reference() all return NULL when no more entries or references exist in the result set to be returned. NULL is also returned if an error occurs while stepping through the entries or references, in which case the error parameters in the session handle ld will be set to indicate the error.

ldap_count_entries() returns the number of entries contained in a chain of entries; if an error occurs such as the res parameter being invalid, -1 is returned. The ldap_count_entries() call can also be used to count the number of entries that remain in a chain if called with a message, entry or reference returned by ldap_first_message(), ldap_next_message(), ldap_first_entry(), ldap_next_entry(), ldap_first_reference(), ldap_next_reference().

ldap_count_references() returns the number of references contained in a chain of search results; if an error occurs such as the res parameter being invalid, -1 is returned. The ldap_count_references() call can also be used to count the number of references that remain in a chain.

ldap_first_attribute

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 ldap_first_entry() or ldap_next_entry().
attr	The attribute whose values are to be retrieved, as returned by ldap_first_attribute() or ldap_next_attribute(), or a caller-supplied string (e.g., "mail").
vals	The values returned by a previous call to ldap_get_values() or ldap_get_values_len().

Usage Notes

Two forms of the various calls are provided. The first form is only suitable for use with non-binary character string data. The second _len form is used with any kind of data.

ldap_get_values() and ldap_get_values_len() return NULL if no values are found for attr or if an error occurs.

ldap_count_values() and ldap_count_values_len() return -1 if an error occurs such as the vals parameter being invalid.

If a NULL vals parameter is passed to ldap_value_free() or ldap_value_free_len(), nothing is done.

Note that the values returned are dynamically allocated and SHOULD be freed by calling either ldap_value_free() or ldap_value_free_len() when no longer in use.

ldap_get_dn

ldap_explode_dn

ldap_explode_rdn

ldap_dn2ufn

ldap_get_dn() is used to retrieve the name of an entry. ldap_explode_dn() and ldap_explode_rdn() are used to break up a name into its component parts. ldap_dn2ufn() is used to convert the name into a more "user friendly" format.

Syntax

```
char *ldap_get_dn( LDAP *ld, LDAPMessage *entry );
char **ldap_explode_dn( const char *dn, int notypes );
char **ldap_explode_rdn( const char *rdn, int notypes );
char *ldap_dn2ufn( const char *dn );
```

Parameters

Table 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 non-zero indicating that the dn or rdn components are to have their type information stripped off (i.e., "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 non-zero 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

```
#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-to-sever communication in the above 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-to-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 3.0.1 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, etc.>
*/
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <netdb.h>
#include "ldap.h"

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

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

usage( s )
char*s;
{
    fprintf( stderr, "usage: %s [options] filter [attributes...]\nwhere:\n", s
);
    fprintf( stderr, "    filter\tRFC-1558 compliant LDAP search filter\n" );
    fprintf( stderr, "    attributes\twhitespace-separated list of attributes to
retrieve\n" );
    fprintf( stderr, "\t\t(if no attribute list is given, all are retrieved)\n"
);
    fprintf( stderr, "options:\n" );
    fprintf( stderr, "    -n\t\tshow what would be done but don't actually
search\n" );
    fprintf( stderr, "    -v\t\trun in verbose mode (diagnostics to standard
```

```
output)\n" );
    fprintf( stderr, "    -t\t\twrite values to files in /tmp\n" );
    fprintf( stderr, "    -u\t\tinclude User Friendly entry names in the
output\n" );
    fprintf( stderr, "    -A\t\tretrieve attribute names only (no values)\n" );
    fprintf( stderr, "    -B\t\tdo not suppress printing of non-ASCII values\n"
);
    fprintf( stderr, "    -L\t\tprint entries in LDIF format (-B is implied)\n"
);
#ifdef LDAP_REFERRALS
    fprintf( stderr, "    -R\t\tdo not automatically follow referrals\n" );
#endif /* LDAP_REFERRALS */
    fprintf( stderr, "    -d level\tset LDAP debugging level to `level'\n" );
    fprintf( stderr, "    -F sep\tprint `sep' instead of `=' between attribute
names and values\n" );
    fprintf( stderr, "    -S attr\tsort the results by attribute `attr'\n" );
    fprintf( stderr, "    -f file\tperform sequence of searches listed in
`file'\n" );
    fprintf( stderr, "    -b basedn\tbase dn for search\n" );
    fprintf( stderr, "    -s scope\tone of base, one, or sub (search scope)\n"
);
    fprintf( stderr, "    -a deref\tone of never, always, search, or find (alias
dereferencing)\n" );
    fprintf( stderr, "    -l time lim\ttime limit (in seconds) for search\n" );
    fprintf( stderr, "    -z size lim\tsize limit (in entries) for search\n" );
    fprintf( stderr, "    -D binddn\tbind dn\n" );
    fprintf( stderr, "    -w passwd\tbind passwd (for simple authentication)\n"
);
#ifdef KERBEROS
    fprintf( stderr, "    -k\t\tuse Kerberos instead of Simple Password
authentication\n" );
#endif
    fprintf( stderr, "    -h host\tldap server\n" );
    fprintf( stderr, "    -p port\tport on ldap server\n" );
    fprintf( stderr, "    -W Wallet\tWallet location\n" );
    fprintf( stderr, "    -P Wpasswd\tWallet Password\n" );
    fprintf( stderr, "    -U SSLAuth\tSSL Authentication Mode\n" );
    return;
}

static char*binddn = LDAPSEARCH_BINDDN;
static char*passwd = NULL;
static char*base = LDAPSEARCH_BASE;
static char*ldaphost = NULL;
static intldapport = LDAP_PORT;
```

```
static char*sep = DEFSEP;
static char*sortattr = NULL;
static intskipsortattr = 0;
static intverbose, not, includeufn, allow_binary, vals2tmp, ldif;
/* TEMP */

main( argc, argv )
intargc;
char**argv;
{
    char*infile, *filtpattern, **attrs, line[ BUFSIZ ];
    FILE*fp;
    intrc, i, first, scope, kerberos, deref, attrsonly;
    intldap_options, timelimit, sizelimit, authmethod;
    LDAP*ld;
    extern char*optarg;
    extern intoptind;
    charlocalHostName[MAXHOSTNAMELEN + 1];
    char *sslwrl = NULL;
    char*sslpasswd = NULL;
int sslauth=0,err=0;

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

    while (( i = getopt( argc, argv,
#ifdef KERBEROS
    "KknuvtRABLD:s:f:h:b:d:p:F:a:w:l:z:S:"
#else
    "nuvtRABLD:s:f:h:b:d:p:F:a:w:l:z:S:W:P:U:"
#endif
    )) != EOF ) {
switch( i ) {
case 'n':/* do Not do any searches */
    ++not;
    break;
case 'v':/* verbose mode */
```

```
    ++verbose;
    break;
case 'd':
#ifdef LDAP_DEBUG
    ldap_debug = lber_debug = atoi( optarg );/* */
#else /* LDAP_DEBUG */
    fprintf( stderr, "compile with -DLDAP_DEBUG for debugging\n" );
#endif /* LDAP_DEBUG */
    break;
#ifdef KERBEROS
case 'k':/* use kerberos bind */
    kerberos = 2;
    break;
case 'K':/* use kerberos bind, 1st part only */
    kerberos = 1;
    break;
#endif
case 'u':/* include UFN */
    ++includeufn;
    break;
case 't':/* write attribute values to /tmp files */
    ++vals2tmp;
    break;
case 'R':/* don't automatically chase referrals */
#ifdef LDAP_REFERRALS
    ldap_options &= ~LDAP_OPT_REFERRALS;
#else /* LDAP_REFERRALS */
    fprintf( stderr,
    "compile with -DLDAP_REFERRALS for referral support\n" );
#endif /* LDAP_REFERRALS */
    break;
case 'A':/* retrieve attribute names only -- no values */
    ++attrsonly;
    break;
case 'L':/* print entries in LDIF format */
    ++ldif;
    /* fall through -- always allow binary when outputting LDIF */
case 'B':/* allow binary values to be printed */
    ++allow_binary;
    break;
case 's':/* search scope */
    if ( strncasecmp( optarg, "base", 4 ) == 0 ) {
scope = LDAP_SCOPE_BASE;
    } else if ( strncasecmp( optarg, "one", 3 ) == 0 ) {
scope = LDAP_SCOPE_ONELEVEL;
```

```
    } else if ( strncasecmp( optarg, "sub", 3 ) == 0 ) {
scope = LDAP_SCOPE_SUBTREE;
    } else {
fprintf( stderr, "scope should be base, one, or sub\n" );
usage( argv[ 0 ] );
                exit(1);
    }
    break;

case 'a':/* set alias deref option */
    if ( strncasecmp( optarg, "never", 5 ) == 0 ) {
deref = LDAP_DEREF_NEVER;
    } else if ( strncasecmp( optarg, "search", 5 ) == 0 ) {
deref = LDAP_DEREF_SEARCHING;
    } else if ( strncasecmp( optarg, "find", 4 ) == 0 ) {
deref = LDAP_DEREF_FINDING;
    } else if ( strncasecmp( optarg, "always", 6 ) == 0 ) {
deref = LDAP_DEREF_ALWAYS;
    } else {
fprintf( stderr, "alias deref should be never, search, find, or always\n" );
usage( argv[ 0 ] );
                exit(1);
    }
    break;

case 'F':/* field separator */
    sep = (char *)strdup( optarg );
    break;
case 'f':/* input file */
    infile = (char *)strdup( optarg );
    break;
case 'h':/* ldap host */
    ldaphost = (char *)strdup( optarg );
    break;
case 'b':/* searchbase */
    base = (char *)strdup( optarg );
    break;
case 'D':/* bind DN */
    binddn = (char *)strdup( optarg );
    break;
case 'p':/* ldap port */
    ldapport = atoi( optarg );
    break;
case 'w':/* bind password */
    passwd = (char *)strdup( optarg );
```

```
    break;
case 'l':/* time limit */
    timelimit = atoi( optarg );
    break;
case 'z':/* size limit */
    sizelimit = atoi( optarg );
    break;
case 'S':/* sort attribute */
    sortattr = (char *)strdup( optarg );
    break;
case 'W':/* Wallet URL */
    sslwrl = (char *)strdup( optarg );
    break;
case 'P':/* Wallet password */
    sslpasswd = (char *)strdup( optarg );
    break;
case 'U':/* SSL Authentication Mode */
    sslauth = atoi( optarg );
    break;
default:
    usage( argv[0] );
            exit(1);
            break;
}
    }

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

```
optind++;
}
        attrs = &argv[ optind ];
    }

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

    if (ldaphost == NULL) {
        if (gethostname(localHostName, MAXHOSTNAMELEN) != 0) {
                perror("gethostname");
                exit(1);
        }
        ldaphost = localHostName;
    }

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

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

    if (sslauth > 1)
    {
       if (!sslwrl || !sslpasswd)
   {
          printf ("Null Wallet or password given\n");
          exit (0);
       }
    }
    if (sslauth > 0)
    {
        if (sslauth == 1)
            sslauth = GSLC_SSL_NO_AUTH;
        else if (sslauth == 2)
            sslauth = GSLC_SSL_ONEWAY_AUTH;
```

```
        else if (sslauth == 3)
            sslauth = GSLC_SSL_TWOWAY_AUTH;
        else
        {
    printf(" Wrong SSL Authenication Mode Value\n");
    exit(0);
        }

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

    ld->ld_deref = deref;
    ld->ld_timelimit = timelimit;
    ld->ld_sizelimit = sizelimit;
    ld->ld_options = ldap_options;

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

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

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

    ldap_unbind( ld );
    exit( rc );
}

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

    sprintf( filter, filtpatt, value );

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

    if ( not ) {
return( LDAP_SUCCESS );
```

```
    }

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

    matches = 0;
    first = 1;
    while ( (rc = ldap_result( ld, LDAP_RES_ANY, sortattr ? 1 : 0, NULL, &res ))
    == LDAP_RES_SEARCH_ENTRY ) {
matches++;
e = ldap_first_entry( ld, res );
if ( !first ) {
    putchar( '\n' );
} else {
    first = 0;
}
print_entry( ld, e, attrsonly );
ldap_msgfree( res );
    }
    if ( rc == -1 ) {
ldap_perror( ld, "ldap_result" );
return( rc );
    }
    if (( rc = ldap_result2error( ld, res, 0 )) != LDAP_SUCCESS ) {
        ldap_perror( ld, "ldap_search" );
    }
    if ( sortattr != NULL ) {
    extern intstrcasecmp();

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

```
    }

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

    ldap_msgfree( res );
    return( rc );
}


print_entry( ld, entry, attrsonly )
    LDAP*ld;
    LDAPMessage*entry;
    intattrsonly;
{
    char*a, *dn, *ufn, tmpfname[ 64 ];
    inti, j, notascii;
    BerElement*ber;
    struct berval**bvals;
    FILE*tmpfp;
    extern char*mktemp();

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

    for ( a = ldap_first_attribute( ld, entry, &ber ); a != NULL;
    a = ldap_next_attribute( ld, entry, ber ) ) {
if ( skipsortattr && strcasecmp( a, sortattr ) == 0 ) {
    continue;
}
```

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

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

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

    if ( ldif ) {
write_ldif_value( a, bvals[ i ]->bv_val,
bvals[ i ]->bv_len );
    } else
{
printf( "%s%s%s\n", a, sep,
```

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


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

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

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

    return( 0 );
}
```

Dependencies and Limitations

This API can work against any release of Oracle Internet Directory. It requires either an Oracle environment or, at minimum, NLS 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).

4

The Oracle Internet Directory PL/SQL API

This chapter introduces the Oracle Internet Directory PL/SQL API and provides examples of how to use it. It contains these topics:

- About the PL/SQL API
- Sample PL/SQL Usage
- Building Applications with PL/SQL LDAP API
- Dependencies and Limitations
- PL/SQL Reference

About the PL/SQL API

The PL/SQL API is packaged in the DBMS_LDAP package. It is based on the C API described in Chapter 3, "The Oracle Internet Directory C API".

You can use the Oracle Internet Directory API release 3.0.1 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.

Sample PL/SQL Usage

This section contains these topics

- Using the PL/SQL API from a Database Trigger
- Using the PL/SQL API for a Search

Using the PL/SQL API 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: `trigger.sql` and `empdata.sql`.

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`

```
$Header: $
Copyright (c) Oracle Corporation 2000. All Rights Reserved.
FILE
trigger.sql
DESCRIPTION
This SQL file creates a database table called 'EMP' and creates a trigger on it
```

```
called LDAP_EMP which will synchronize all changes happening to the table with
an LDAP server. The changes to the database table are reflected/replicated to
the LDAP directory using the DBMS_LDAP package.
This script assumes the following:
LDAP server hostname: NULL (local host)
LDAP server portnumber: 389
Directory container for employee records: o=acme, dc=com
Username/Password for Directory Updates: cn=orcladmin/welcome
The aforementioned variables could be customized for different environments by
changing the appropriate variables in the code below.
Table Definition:
Employee Details(Columns) in Database Table(EMP):
EMP_ID          Number
FIRST_NAME      Varchar2
LAST_NAME       Varchar2
MANAGER_ID      Number
PHONE_NUMBER    Varchar2
MOBILE          Varchar2
ROOM_NUMBER     Varchar2
TITLE           Varchar2

LDAP Schema Definition & mapping to relational schema EMP:
Corresponding Data representation in LDAP directory:
    DN              cn=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

MODIFIED   (MM/DD/YY)
rbollu 07/21/00 - created

—Creating EMP table

PROMPT Dropping Table EMP ..
drop table EMP;
```

```
PROMPT Creating Table EMP ..
CREATE TABLE EMP (
    EMP_ID    NUMBER,             Employee Number
    FIRST_NAME  VARCHAR2(256),    First Name
    LAST_NAME  VARCHAR2(256),     Last Name
    MANAGER_ID  NUMBER,           Manager Number
    PHONE_NUMBER VARCHAR2(256),   Telephone Number
    MOBILE   VARCHAR2(256),       Mobile Number
    ROOM_NUMBER  VARCHAR2(256),   Room Number
    TITLE   VARCHAR2(256)         Title in the company
);

—Creating Trigger LDAP_EMP

PROMPT Creating Trigger LDAP_EMP ..

CREATE OR REPLACE TRIGGER LDAP_EMP
AFTER INSERT OR DELETE OR UPDATE ON EMP
FOR EACH ROW

DECLARE
    retval   PLS_INTEGER;
    emp_session  DBMS_LDAP.session;
    emp_dn   VARCHAR2(256);
    emp_rdn   VARCHAR2(256);
    emp_array  DBMS_LDAP.MOD_ARRAY;
    emp_vals  DBMS_LDAP.STRING_COLLECTION ;
    ldap_host  VARCHAR2(256);
    ldap_port  VARCHAR2(256);
    ldap_user  VARCHAR2(256);
    ldap_passwd  VARCHAR2(256);
    ldap_base  VARCHAR2(256);
BEGIN

    retval       := -1;
    -- Customize the following variables as needed
    ldap_host  := NULL;
    ldap_port  := '389';
    ldap_user  := 'cn=orcladmin';
    ldap_passwd:= 'welcome';
    ldap_base  := 'o=acme,dc=com';
    -- end of customizable settings

    DBMS_OUTPUT.PUT('Trigger [LDAP_EMP]: Replicating changes ');
    DBMS_OUTPUT.PUT_LINE('to directory .. ');
```

```
DBMS_OUTPUT.PUT_LINE(RPAD('LDAP Host ',25,' ') || ': ' || ldap_host);
DBMS_OUTPUT.PUT_LINE(RPAD('LDAP Port ',25,' ') || ': ' || ldap_port);

-- Choosing exceptions to be raised by DBMS_LDAP library.
DBMS_LDAP.USE_EXCEPTION := TRUE;

-- Initialize ldap library and get session handle.
emp_session := DBMS_LDAP.init(ldap_host,ldap_port);

DBMS_OUTPUT.PUT_LINE (RPAD('Ldap session ',25,' ')  || ': ' ||
    RAWTOHEX(SUBSTR(emp_session,1,8)) ||
    '(returned from init)');

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

    DBMS_OUTPUT.PUT_LINE(RPAD('simple_bind_s Returns ',25,' ') || ': '
          || TO_CHAR(retval));

-- Process New Entry in the database

IF INSERTING THEN

    -- Create and setup attribute array for the New entry
    emp_array := DBMS_LDAP.create_mod_array(14);

    -- RDN to be - cn="FIRST_NAME LAST_NAME"

    emp_vals(1) := :new.FIRST_NAME || ' ' || :new.LAST_NAME;

    DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_ADD,
    'cn',emp_vals);

    emp_vals(1) := :new.LAST_NAME;

    DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_ADD,
                'sn',emp_vals);

    emp_vals(1) := :new.FIRST_NAME;

    DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_ADD,
                'givenname',emp_vals);

    emp_vals(1) := 'top';
```

```
emp_vals(2) := 'person';
emp_vals(3) := 'organizationalPerson';
emp_vals(4) := 'inetOrgPerson';

DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_ADD,
            'objectclass',emp_vals);

emp_vals.DELETE;
emp_vals(1) := :new.PHONE_NUMBER;

DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_ADD,
            'telephonenumber',emp_vals);

emp_vals(1) := :new.MOBILE;

DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_ADD,
            'mobile',emp_vals);

emp_vals(1) := :new.ROOM_NUMBER;

DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_ADD,
            'roomNumber',emp_vals);

emp_vals(1) := :new.TITLE;

DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_ADD,
            'title',emp_vals);

emp_vals(1) := :new.EMP_ID;

DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_ADD,
            'employeeNumber',emp_vals);

emp_vals(1) := :new.FIRST_NAME;

DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_ADD,
            'userpassword',emp_vals);

-- DN for Entry to be Added under 'ldap_base' [o=acme, dc=com]

emp_dn := 'cn=' || :new.FIRST_NAME || ' ' ||
:new.LAST_NAME || ', ' || ldap_base ;
DBMS_OUTPUT.PUT_LINE(RPAD('Adding Entry for DN ',25,' ') || ': ['
           || emp_dn || ']');
```

```
    -- Add new Entry to ldap directory
    retval := DBMS_LDAP.add_s(emp_session,emp_dn,emp_array);
    DBMS_OUTPUT.PUT_LINE(RPAD('add_s Returns ',25,' ')  || ': '
              || TO_CHAR(retval));

    -- Free attribute array (emp_array)
    DBMS_LDAP.free_mod_array(emp_array);

END IF; -- INSERTING

-- Process Entry deletion in database

IF DELETING THEN

-- DN for Entry to be deleted under 'ldap_base' [o=acme, dc=com]

emp_dn := 'cn=' || :old.FIRST_NAME || ' ' ||
:old.LAST_NAME || ', ' || ldap_base ;
DBMS_OUTPUT.PUT_LINE(RPAD('Deleting Entry for DN ',25,' ') ||
         ': [' || emp_dn || ']');

-- Delete entry in ldap directory
retval := DBMS_LDAP.delete_s(emp_session,emp_dn);
     DBMS_OUTPUT.PUT_LINE(RPAD('delete_s Returns ',25,' ') || ': ' ||
           TO_CHAR(retval));

END IF; -- DELETING

-- Process updated Entry in database

IF UPDATING THEN

    -- Since two Table columns(in this case) constitue a RDN
    -- check for any changes and update RDN in ldap directory
    -- before updating any other attributes of the Entry.

    IF :old.FIRST_NAME <> :new.FIRST_NAME OR
         :old.LAST_NAME  <> :new.LAST_NAME THEN

        emp_dn := 'cn=' || :old.FIRST_NAME || ' ' ||
            :old.LAST_NAME || ', ' || ldap_base;

        emp_rdn := 'cn=' || :new.FIRST_NAME || ' ' || :new.LAST_NAME;

        DBMS_OUTPUT.PUT_LINE(RPAD('Renaming OLD DN ',25,' ') ||
```

```
                ': [' || emp_dn || ']');
    DBMS_OUTPUT.PUT_LINE(RPAD(' => NEW RDN ',25,' ') ||
                ': [' || emp_rdn || ']' );
    retval := DBMS_LDAP.modrdn2_s(emp_session,emp_dn,emp_rdn,
                DBMS_LDAP.MOD_DELETE);
    DBMS_OUTPUT.PUT_LINE(RPAD('modrdn2_s Returns ',25,' ') || ': ' ||
                TO_CHAR(retval));
END IF;

-- DN for Entry to be updated under 'ldap_base' [o=acme, dc=com]

emp_dn := 'cn=' || :new.FIRST_NAME || ' ' ||
   :new.LAST_NAME || ', ' || ldap_base;

DBMS_OUTPUT.PUT_LINE(RPAD('Updating Entry for DN ',25,' ') ||
            ': [' || emp_dn || ']');

-- Create and setup attribute array(emp_array) for updated entry
emp_array := DBMS_LDAP.create_mod_array(7);

emp_vals(1) := :new.LAST_NAME;

DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_REPLACE,
            'sn',emp_vals);

emp_vals(1) := :new.FIRST_NAME;

DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_REPLACE,
            'givenname',emp_vals);

emp_vals(1) := :new.PHONE_NUMBER;

DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_REPLACE,
            'telephonenumber',emp_vals);

emp_vals(1) := :new.MOBILE;

DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_REPLACE,
            'mobile',emp_vals);

emp_vals(1) := :new.ROOM_NUMBER;

DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_REPLACE,
            'roomNumber',emp_vals);
```

```
        emp_vals(1) := :new.TITLE;

        DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_REPLACE,
                    'title',emp_vals);

        emp_vals(1) := :new.EMP_ID;

        DBMS_LDAP.populate_mod_array(emp_array,DBMS_LDAP.MOD_REPLACE,
                    'employeeNumber',emp_vals);

        -- Modify entry in ldap directory
          retval := DBMS_LDAP.modify_s(emp_session,emp_dn,emp_array);

           DBMS_OUTPUT.PUT_LINE(RPAD('modify_s Returns ',25,' ') || ': ' ||
                       TO_CHAR(retval));

        -- Free attribute array (emp_array)
        DBMS_LDAP.free_mod_array(emp_array);

    END IF; -- UPDATING

    -- Unbind from ldap directory
    retval := DBMS_LDAP.unbind_s(emp_session);

    DBMS_OUTPUT.PUT_LINE(RPAD('unbind_res Returns ',25,' ') ||  ': ' ||
               TO_CHAR(retval));

    DBMS_OUTPUT.PUT_LINE('Directory operation Successful .. exiting');

    -- Handle Exceptions
    EXCEPTION
        WHEN OTHERS THEN
        -- TODO : should the trigger call unbind at this point ??
        --   what if the exception was raised from unbind itself ??

        DBMS_OUTPUT.PUT_LINE(' Error code    : ' || TO_CHAR(SQLCODE));
        DBMS_OUTPUT.PUT_LINE(' Error Message : ' || SQLERRM);
        DBMS_OUTPUT.PUT_LINE(' Exception encountered .. exiting');

END;
/
------------------------------END OF trigger.sql--------------------------
```

Using the PL/SQL API 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.

```
$Header: $

Copyright (c) Oracle Corporation 2000. All Rights Reserved.

FILE
    search.sql

DESCRIPTION

    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.



MODIFIED    (MM/DD/YY)
akolli07/21/00 - created

set serveroutput on size 30000

DECLARE
    retval       PLS_INTEGER;
    my_session   DBMS_LDAP.session;
    my_attrs     DBMS_LDAP.string_collection;
    my_message   DBMS_LDAP.message;
    my_entry     DBMS_LDAP.message;
```

```
    entry_index  PLS_INTEGER;
    my_dn        VARCHAR2(256);
    my_attr_name VARCHAR2(256);
    my_ber_elmt  DBMS_LDAP.ber_element;
    attr_index   PLS_INTEGER;
    i          PLS_INTEGER;
    my_vals DBMS_LDAP.STRING_COLLECTION ;
    ldap_host  VARCHAR2(256);
    ldap_port  VARCHAR2(256);
    ldap_user  VARCHAR2(256);
    ldap_passwd  VARCHAR2(256);
    ldap_base  VARCHAR2(256);

BEGIN
    retval         := -1;

    -- Please customize the following variables as needed
    ldap_host  := NULL ;
    ldap_port  := '389';
    ldap_user  := 'cn=orcladmin';
    ldap_passwd:= 'welcome';
    ldap_base  := 'o=acme,dc=com';
    -- end of customizable settings

     DBMS_OUTPUT.PUT('DBMS_LDAP Search Example ');
     DBMS_OUTPUT.PUT_LINE('to directory .. ');
     DBMS_OUTPUT.PUT_LINE(RPAD('LDAP Host ',25,' ') || ': ' || ldap_host);
     DBMS_OUTPUT.PUT_LINE(RPAD('LDAP Port ',25,' ') || ': ' || ldap_port);

     -- Choosing exceptions to be raised by DBMS_LDAP library.
     DBMS_LDAP.USE_EXCEPTION := TRUE;


     my_session := DBMS_LDAP.init(ldap_host,ldap_port);

     DBMS_OUTPUT.PUT_LINE (RPAD('Ldap session ',25,' ')  || ': ' ||
         RAWTOHEX(SUBSTR(my_session,1,8)) ||
         '(returned from init)');

     -- bind to the directory
     retval := DBMS_LDAP.simple_bind_s(my_session,
                ldap_user, ldap_passwd);

     DBMS_OUTPUT.PUT_LINE(RPAD('simple_bind_s Returns ',25,' ') || ': '
              || TO_CHAR(retval));
```

```
-- issue the search
my_attrs(1) := '*'; -- retrieve all attributes
retval := DBMS_LDAP.search_s(my_session, ldap_base,
                             DBMS_LDAP.SCOPE_SUBTREE,
                             'objectclass=*',
                             my_attrs,
                             0,
                             my_message);

DBMS_OUTPUT.PUT_LINE(RPAD('search_s Returns ',25,' ') || ': '
        || TO_CHAR(retval));
DBMS_OUTPUT.PUT_LINE (RPAD('LDAP message  ',25,' ')  || ': ' ||
    RAWTOHEX(SUBSTR(my_message,1,8)) ||
    '(returned from search_s)');

-- count the number of entries returned
retval := DBMS_LDAP.count_entries(my_session, my_message);
DBMS_OUTPUT.PUT_LINE(RPAD('Number of Entries ',25,' ') || ': '
        || TO_CHAR(retval));
DBMS_OUTPUT.PUT_
LINE('--------------------------------------------------');


-- get the first entry
my_entry := DBMS_LDAP.first_entry(my_session, my_message);
entry_index := 1;

-- Loop through each of the entries one by one
while my_entry IS NOT NULL loop
      -- print the current entry
      my_dn := DBMS_LDAP.get_dn(my_session, my_entry);
      -- DBMS_OUTPUT.PUT_LINE ('        entry #' || TO_CHAR(entry_index) ||
       -- ' entry ptr: ' || RAWTOHEX(SUBSTR(my_entry,1,8)));
      DBMS_OUTPUT.PUT_LINE ('        dn: ' || my_dn);
      my_attr_name := DBMS_LDAP.first_attribute(my_session,my_entry,
      my_ber_elmt);
      attr_index := 1;
      while my_attr_name IS NOT NULL loop
            my_vals := DBMS_LDAP.get_values (my_session, my_entry,
            my_attr_name);
            if my_vals.COUNT > 0 then
              FOR i in my_vals.FIRST..my_vals.LAST loop
                DBMS_OUTPUT.PUT_LINE('           ' || my_attr_name || ' : '
       ||
```

```
                    SUBSTR(my_vals(i),1,200));
                  end loop;
            end if;
            my_attr_name := DBMS_LDAP.next_attribute(my_session,my_entry,
            my_ber_elmt);
            attr_index := attr_index+1;
      end loop;
      my_entry := DBMS_LDAP.next_entry(my_session, my_entry);
      DBMS_OUTPUT.PUT_
   LINE('================================================');
      entry_index := entry_index+1;
end loop;

-- unbind from the directory
retval := DBMS_LDAP.unbind_s(my_session);
DBMS_OUTPUT.PUT_LINE(RPAD('unbind_res Returns ',25,' ') ||  ': ' ||
          TO_CHAR(retval));

DBMS_OUTPUT.PUT_LINE('Directory operation Successful .. exiting');

-- Handle Exceptions
   EXCEPTION
       WHEN OTHERS THEN
             DBMS_OUTPUT.PUT_LINE(' Error code    : ' || TO_CHAR(SQLCODE));
             DBMS_OUTPUT.PUT_LINE(' Error Message : ' || SQLERRM);
             DBMS_OUTPUT.PUT_LINE(' Exception encountered .. exiting');
END;
/
------------------------------END OF trigger.sql--------------------------
```

Building Applications with PL/SQL LDAP API

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.

PL/SQL Reference

The PL/SQL package 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

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 DBMS_LDAP API Subprograms

Function or Procedure	Description
FUNCTION bind_s	The function bind_s can be used to perform complex authentication to the directory server.
FUNCTION unbind_s	The function unbind_s is used for closing an active LDAP session.
FUNCTION compare_s	The function compare_s can be used to test if a particular attribute in a particular entry has a particular value.
FUNCTION search_s	The function search_s performs a synchronous search in the LDAP server. It returns control to the PL/SQL environment only after all of the search results have been sent by the server or if the search request is 'timed-out' by the server.
FUNCTION search_st	The function search_st performs a synchronous search in the LDAP server with a client side time-out. It returns control to the PL/SQL environment only after all of the search results have been sent by the server or if the search request is 'timed-out' by the client or the server.
FUNCTION first_entry	The function first_entry is used to retrieve the first entry in the result set returned by either search_s or search_st.
FUNCTION next_entry	The function next_entry() is used to iterate to the next entry in the result set of a search operation.
FUNCTION count_entries	This function is used to count the number of entries in the result set. It can also be used to count the number of entries remaining during a traversal of the result set using a combination of the functions first_entry() and next_entry().
FUNCTION first_attribute	The function first_attribute() fetches the first attribute of a given entry in the result set.
FUNCTION next_attribute	The function next_attribute() fetches the next attribute of a given entry in the result set.
FUNCTION get_dn	The function get_dn() retrieves the X.500 distinguished name of given entry in the result set.
FUNCTION get_values	The function get_values() can be used to retrieve all of the values associated for a given attribute in a given entry.

Table 4–1 DBMS_LDAP API Subprograms

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

Table 4–1 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.

Exception Summary

The DBMS_LDAP package shipped with Oracle9*i* release 9.0.1 can generate the following exceptions:

Table 4–2 DBMS_LDAP Exception Summary

Exception Name	Oracle Error Number	Cause of Exception
general_error	31202	Raised anytime an error is encountered that does not have a specific PL / SQL exception associated with it. The error string contains the description of the problem in the local language of the user.
init_failed	31203	Raised by DBMS_LDAP.init() if there are some problems.
invalid_session	31204	Raised by all functions and procedures in the DBMS_LDAP package if they are passed an invalid session handle.
invalid_auth_method	31205	Raised by DBMS_LDAP.bind_s() if the authentication method requested is not supported.
invalid_search_scope	31206	Raised by all of the 'search' functions if the scope of the search is invalid.
invalid_search_time_val	31207	Raised by time based search function: DBMS_ LDAP.search_st() if it is given an invalid value for the time limit.
invalid_message	31208	Raised by all functions that iterate through a result-set for getting entries from a search operation if the message handle given to them is invalid.

Table 4–2 DBMS_LDAP Exception Summary

Exception Name	Oracle Error Number	Cause of Exception
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.
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
ld	A valid LDAP session handle.
dn	The Distinguished Name of the User that we are trying to login as.
passwd	A text string containing the password.

Return Values

Table 4–8 SIMPLE_BIND_S Function Return Values

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

Exceptions

Table 4–9 SIMPLE_BIND_S Function Exceptions

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

Usage Notes

DBMS_LDAP.simple_bind_s() can be used to authenticate a user whose directory distinguished name and directory password are known. It can be called only after a valid LDAP session handle is obtained from a call to DBMS_LDAP.init().

FUNCTION bind_s

The function bind_s can be used to perform complex authentication to the directory server.

Syntax

```
FUNCTION  bind_s
(
    ld     IN SESSION,
    dn     IN VARCHAR2,
    cred IN VARCHAR2,
    meth IN PLS_INTEGER
)
    RETURN PLS_INTEGER;
```

Parameters

Table 4–10 BIND_S Function Parameters

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

Return Values

Table 4–11 BIND_S Function Return Values

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

Exceptions

Table 4–12 BIND_S Function Exceptions

Exception	Description
invalid_session	Raised if the session handle `ld` is invalid.
invalid_auth_method	Raised if the authentication method requested is not supported.
general_error	For all other errors. The error string associated with this exception will explain the error in detail.

Usage Notes

DBMS_LDAP.bind_s() can be used to authenticate a user. It can be called only after a valid LDAP session handle is obtained from a call to DBMS_LDAP.init().

See Also

DBMS_LDAP.init(), DBMS_LDAP.simple_bind_s().

FUNCTION unbind_s

The function unbind_s is used for closing an active LDAP session.

Syntax

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

Parameters

Table 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 `ld` is invalid.
general_error	For all other errors. The error string associated with this exception will explain the error in detail.

Usage Notes

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

See Also

DBMS_LDAP.bind_s()

FUNCTION search_s

The function search_s performs a synchronous search in the LDAP server. It returns control to the PL/SQL environment only after all of the search results have been sent by the server or if the search request is 'timed-out' by the server.

Syntax

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

Parameters

Table 4–19 SEARCH_S Function Parameters

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

Table 4–19 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 `ld` is invalid.
invalid_search_scope	Raised if the search scope is not one of SCOPE_BASE, SCOPE_ONELEVEL, or SCOPE_SUBTREE.
general_error	For all other errors. The error string associated with this exception will explain the error in detail.

Usage Notes

The function search_s() issues a search operation and does not return control to the user environment until all of the results have been returned from the server. Entries returned from the search (if any) are contained in the res parameter. This parameter is opaque to the caller. Entries, attributes, values, etc., can be extracted by calling the parsing routines described below.

See Also

DBMS_LDAP.search_st(), DBMS_LDAP.first_entry(), DBMS_LDAP.next_entry.

FUNCTION search_st

The function search_st performs a synchronous search in the LDAP server with a client-side time-out. It returns control to the PL/SQL environment only after all of the search results have been sent by the server or if the search request is 'timed-out' by the client or the server.

Syntax

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

Parameters

Table 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 SEARCH_ST Function Parameters

Parameter	Description
attrsonly	A boolean value that MUST be zero if both attribute types and values are to be returned, and non-zero if only types are wanted.
tv	The time-out value expressed in seconds and microseconds that should be used for this search.
res	This is a result parameter which will contain the results of the search upon completion of the call. If no results are returned, *res is set to NULL.

Return Values

Table 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 `ld` is invalid.
invalid_search_scope	Raised if the search scope is not one of SCOPE_BASE, SCOPE_ONELEVEL or SCOPE_SUBTREE.
invalid_search_time_value	Raised if the time value specified for the time-out is invalid.
general_error	For all other errors. The error string associated with this exception will explain the error in detail.

Usage Notes

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

See Also

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

FUNCTION first_entry

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

Syntax

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

Parameters

Table 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, ld is invalid.
invalid_message	Raised if the incoming 'msg' handle is invalid.

Usage Notes

The function next_entry() should always be called after a call to the function first_entry(). Also, the return value of a successful call to next_entry() should be used as 'msg' argument used in a subsequent call to the function next_entry() to fetch the next entry in the list.

See Also

DBMS_LDAP.first_entry(), DBMS_LDAP.search_s(), DBMS_LDAP.search_st()

FUNCTION count_entries

This function is used to count the number of entries in the result set. It can also be used to count the number of entries remaining during a traversal of the result set using a combination of the functions first_entry() and next_entry().

Syntax

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

Parameters

Table 4–31 COUNT_ENTRY Function Parameters

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

Return Values

Table 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
invalid_session	Raised if the session handle `ld` is invalid.
invalid_message	Raised if the incoming 'msg' handle is invalid.
count_entry_error	Raised if there was a problem in counting the entries.

Usage Notes

`count_entries()` returns the number of entries contained in a chain of entries; if an error occurs such as the res parameter being invalid, -1 is returned. The `count_entries()` call can also be used to count the number of entries that remain in a chain if called with a message, entry or reference returned by `first_message()`, `next_message()`, `first_entry()`, `next_entry()`, `first_reference()`, `next_reference()`.

See Also

DBMS_LDAP.first_entry(), DBMS_LDAP.next_entry().

FUNCTION first_attribute

The function first_attribute() fetches the first attribute of a given entry in the result set.

Syntax

```
FUNCTION first_attribute
(
    ld       IN  SESSION,
    msg      IN  MESSAGE,
    ber_elem OUT BER_ELEMENT
)
    RETURN VARCHAR2;
```

Parameters

Table 4–34 FIRST_ATTRIBUTE Function Parameter

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

Return Values

Table 4–35 FIRST_ATTRIBUTE Function Return Values

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

Exceptions

Table 4–36 FIRST_ATTRIBUTE Function Exceptions

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

Table 4–36 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,
    msg      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.
msg	The entry whose attributes are to be stepped through, as returned by `first_entry()` or `next_entry()`.
ber_elem	A handle to a BER ELEMENT that is used to keep track of which attribute in the entry has been read.

Return Values

Table 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 `ld` is invalid.
invalid_message	Raised if the incoming 'msg' handle is invalid.

Usage Notes

The handle to the BER_ELEMENT returned as a function parameter to first_attribute() should be used in the next call to next_attribute() to iterate through the various attributes of an entry. The name of the attribute returned from a call to next_attribute() can in turn be used in calls to the functions get_values() or get_values_len() to get the values of that particular attribute.

See Also

DBMS_LDAP.first_attribute(), DBMS_LDAP.get_values(), DBMS_LDAP.get_values_len(), DBMS_LDAP.first_entry(), DBMS_LDAP.next_entry().

FUNCTION get_dn

The function get_dn() retrieves the X.500 distinguished name of given entry in the result set.

Syntax

```
FUNCTION get_dn
(
    ld  IN SESSION,
    msg IN MESSAGE
)
    RETURN VARCHAR2;
```

Parameters

Table 4–40 GET_DN Function Parameters

Parameter	Description
ld	A valid LDAP session handle.
msg	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 `ld` is invalid.
invalid_message	Raised if the incoming 'msg' handle is invalid.
get_dn_error	Raised if there was a problem in determining the DN

Usage Notes

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

See Also

DBMS_LDAP.explode_dn().

FUNCTION get_values

The function get_values() can be used to retrieve all of the values associated 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
ld	A valid LDAP session handle
ldapentry	A valid handle to an entry returned from a search result
attr	The name of the attribute for which values are being sought

Return Values

Table 4–44 GET_VALUES Function Return Values

Value	Description
STRING_COLLECTION (function return)	A PL/SQL string collection containing all of the values of the given attribute
	NULL if there are no values associated with the given attribute

Exceptions

Table 4–45 GET_VALUES Function Exceptions

Exception	Description
invalid session	Raised if the session handle `ld` is invalid.
invalid message	Raised if the incoming 'entry handle' is invalid.

Usage Notes

The function get_values() can only be called after the handle to entry has been first retrieved by call to either first_entry() or next_entry(). The name of the attribute may be known beforehand or can also be determined by a call to first_attribute() or next_attribute().The function get_values() always assumes that the data-type of the attribute it is retrieving is 'String'. For retrieving binary data-types, get_values_len() should be used.

See Also

DBMS_LDAP.first_entry(), DBMS_LDAP.next_entry(), DBMS_LDAP.count_values(), DBMS_LDAP.get_values_len().

FUNCTION get_values_len

The function get_values_len() can be used to retrieve values of attributes that have a 'Binary' syntax.

Syntax

```
FUNCTION get_values_len
(
    ld   IN SESSION,
    ldapentry IN MESSAGE,
    attr IN VARCHAR2
)
    RETURN BINVAL_COLLECTION;
```

Parameters

Table 4–46 GET_VALUES_LEN Function Parameters

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

Return Values

Table 4–47 GET_VALUES_LEN Function Return Values

Value	Description
BINVAL_COLLECTION (function return	A PL/SQL 'Raw' collection containing all the values of the given attribute. NULL if there are no values associated with the given attribute.

Exceptions

Table 4–48 GET_VALUES_LEN Function Exceptions

Exception	Description
invalid_session	Raised if the session handle `ld` is invalid.
invalid_message	Raised if the incoming 'entry handle' is invalid

Usage Notes

The function get_values_len() can only be called after the handle to entry has been first retrieved by call to either first_entry() or next_entry().The name of the attribute may be known beforehand or can also be determined by a call to first_attribute() or next_attribute().This function can be used to retrieve both binary and non-binary attribute values.

See Also

DBMS_LDAP.first_entry(), DBMS_LDAP.next_entry(), DBMS_LDAP.count_values_len(), DBMS_LDAP.get_values().

FUNCTION delete_s

The function delete_s() can be used to remove a leaf entry in the LDAP Directory Information Tree.

Syntax

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

Parameters

Table 4–49 DELETE_S Function Parameters

Parameter Name	Description
ld	A valid LDAP session
entrydn	The X.500 distinguished name of the entry to delete

Return Values

Table 4–50 DELETE_S Function Return Values

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

Exceptions

Table 4–51 DELETE_S Function Exceptions

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

Usage Notes

The function delete_s() can be used to remove only leaf 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
ld	A valid LDAP session handle.
entrydn	The distinguished name of the entry (This entry must be a leaf node in the DIT.).
newrdn	The new relative distinguished name of the entry.
deleteoldrdn	A boolean value that if non-zero indicates that the attribute values from the old name should be removed from the entry.

Return Values

Table 4–53 MODRDN2_S Function Return Values

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

Exceptions

Table 4–54 MODRDN2_S Function Exceptions

Exception	Description
invalid_session	Raised if the session handle `ld` is invalid.
invalid_entry_dn	Raised if the distinguished name of the entry is invalid.
invalid_rdn	Invalid LDAP RDN.
invalid_deleteoldrdn	Invalid LDAP deleteoldrdn.
general error	For all other errors. The error string associated with this exception will explain the error in detail.

Usage Notes

The function nodrdn2_s() can be used to rename the leaf nodes of a DIT. It simply changes the relative distinguished name by which they are known. The use of this function is being deprecated in the LDAP v3 standard. Please use rename_s() which can achieve the same foundation.

See Also

DBMS_LDAP.rename_s().

FUNCTION err2string

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

Syntax

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

Parameters

Table 4–55 ERR2STRING Function Parameters

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

Return Values

Table 4–56 ERR2STRING Function Return Values

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

Exceptions

Table 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,
    modval   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
modval	This field specifies the attribute values to add, delete, or replace. It is for the binary values

Return Values

Table 4–65 POPULATE_MOD_ARRAY (Binary Version) Procedure Return Values

Value	Description
N / A	

Exceptions

Table 4–66 POPULATE_MOD_ARRAY (Binary Version) Procedure Exceptions

Exception	Description
invalid_mod_array	Invalid LDAP mod array
invalid_mod_option	Invalid LDAP mod option
invalid_mod_type	Invalid LDAP mod type
invalid_mod_value	Invalid LDAP mod value

Usage Notes

This function is one of the preparation steps for `DBMS_LDAP.add_s` and `DBMS_LDAP.modify_s`. It has to happen after `DBMS_LDAP.create_mod_array` called.

See Also

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

FUNCTION modify_s

Performs a synchronous modification of an existing LDAP directory entry.

Syntax

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

Parameters

Table 4–67 MODIFY_S Function Parameters

Parameter	Description
ld	This parameter is a handle to an LDAP session, as returned by a successful call to `DBMS_LDAP.init()`.
entrydn	This parameter specifies the name of the directory entry whose contents are to be modified.
modptr	This parameter is the handle to an LDAP mod structure, as returned by successful call to `DBMS_LDAP.create_mod_array()`.

Return Values

Table 4–68 MODIFY_S Function Return Values

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

Exceptions

Table 4–69 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 4–70 ADD_S Function Parameters

Parameter	Description
ld	This parameter is a handle to an LDAP session, as returned by a successful call to `DBMS_LDAP.init()`.
entrydn	This parameter specifies the name of the directory entry to be created.
modptr	This parameter is the handle to an LDAP mod structure, as returned by successful call to `DBMS_LDAP.create_mod_array()`.

Return Values

Table 4–71 ADD_S Function Return Values

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

Exceptions

Table 4–72 ADD_S Function Exceptions

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

Usage Notes

The parent entry of the entry to be added must already exist in the directory. This function call has to follow successful calls of `DBMS_LDAP.create_mod_array()` and `DBMS_LDAP.populate_mod_array()`.

See Also

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

PROCEDURE free_mod_array

Frees the memory allocated by DBMS_LDAP.create_mod_array().

Syntax

```
PROCEDURE free_mod_array
(
    modptr IN DBMS_LDAP.MOD_ARRAY
);
```

Parameters

Table 4–73 FREE_MOD_ARRAY Procedure Parameters

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

Return Values

Table 4–74 FREE_MOD_ARRAY Procedure Return Value

Value	Description
N / A	

Exceptions

Table 4–75 FREE_MOD_ARRAY Procedure Exceptions

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

Usage Notes

N / A

See Also

DBMS_LDAP.populate_mod_array(), DBMS_LDAP.modify_s(), DBMS_LDAP.add_s(), and DBMS_LDAP.create_mod_array().

FUNCTION count_values

Counts the number of values returned by `DBMS_LDAP.get_values()`.

Syntax

```
FUNCTION count_values
(
    values IN DBMS_LDAP.STRING_COLLECTION
)
    RETURN PLS_INTEGER;
```

Parameters

Table 4–76 COUNT_VALUES Function Parameters

Parameter	Description
values	The collection of string values.

Return Values

Table 4–77 COUNT_VALUES Function Return Values

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

Exceptions

Table 4–78 COUNT_VALUES Function Exceptions

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

Usage Notes

N / A

See Also

`DBMS_LDAP.count_values_len()`, `DBMS_LDAP.get_values()`.

FUNCTION count_values_len

Counts the number of values returned by DBMS_LDAP.get_values_len().

Syntax

```
FUNCTION count_values_len
(
    values IN DBMS_LDAP.BINVAL_COLLECTION
)
    RETURN PLS_INTEGER;
```

Parameters

Table 4–79 COUNT_VALUES_LEN Function Parameters

Parameter	Description
values	The collection of binary values.

Return Values

Table 4–80 COUNT_VALUES_LEN Function Return Values

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

Exceptions

Table 4–81 COUNT_VALUES_LEN Function Exceptions

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

Usage Notes

N / A

See Also

DBMS_LDAP.count_values(), DBMS_LDAP.get_values_len().

FUNCTION rename_s

Renames an LDAP entry synchronously.

Syntax

```
FUNCTION rename_s
(
    ld           IN SESSION,
    dn           IN VARCHAR2,
    newrdn       IN VARCHAR2,
    newparent    IN VARCHAR2,
    deleteoldrdn IN PLS_INTEGER,
    serverctrls  IN LDAPCONTROL,
    clientctrls  IN LDAPCONTROL
)
    RETURN PLS_INTEGER;
```

Parameters

Table 4–82 RENAME_S Function Parameters

Parameter	Description
ld	This parameter is a handle to an LDAP session, as returned by a successful call to `DBMS_LDAP.init()`.
dn	This parameter specifies the name of the directory entry to be renamed or moved.
newrdn	This parameter specifies the new RDN.
newparent	This parameter specifies the DN of the new parent.
deleteoldrdn	This parameter specifies if the old RDN should be retained. If this value is 1, then the old RDN will be removed.
serverctrls	Currently not supported.
clientctrls	Currently not supported.

Return Values

Table 4–83 RENAME_S Function Return Values

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

Exceptions

Table 4–84 RENAME_S Function Exceptions

Exception	Description
invalid_session	Invalid LDAP Session.
invalid_entry_dn	Invalid LDAP DN.
invalid_rdn	Invalid LDAP RDN.
invalid_newparent	Invalid LDAP newparent.
invalid_deleteoldrdn	Invalid LDAP deleteoldrdn.

Usage Notes

N / A

See Also

`DBMS_LDAP.modrdn2_s()`.

FUNCTION explode_dn

Breaks a DN up into its components.

Syntax

```
FUNCTION explode_dn
(
    dn       IN VARCHAR2,
    notypes IN PLS_INTEGER
)
    RETURN STRING_COLLECTION;
```

Parameters

Table 4–85 EXPLODE_DN Function Parameters

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

Return Values

Table 4–86 EXPLODE_DN Function Return Values

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

Exceptions

Table 4–87 EXPLODE_DN Function Exceptions

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

Usage Notes

N/A

See Also

`DBMS_LDAP.get_dn()`.

FUNCTION open_ssl

Establishes an SSL (Secure Sockets Layer) connection over an existing LDAP connection.

Syntax

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

Parameters

Table 4–88 OPEN_SSL Function Parameters

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

Return Values

Table 4–89 OPEN_SSL Function Return Values

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

Exceptions

Table 4–90 OPEN_SSL Function Exceptions

Exception	Description
invalid_session	Invalid LDAP Session.
invalid_ssl_wallet_loc	Invalid LDAP SSL wallet location.
invalid_ssl_wallet_ passwd	Invalid LDAP SSL wallet passwd.
invalid_ssl_auth_mode	Invalid LDAP SSL authentication mode.

Usage Notes

Need to call `DBMS_LDAP.init()` first to acquire a valid ldap session.

See Also

`DBMS_LDAP.init()`.

FUNCTION msgfree

This function frees the chain of messages associated with the message handle returned by synchronous search functions.

Syntax

```
FUNCTION msgfree
(
    res             IN MESSAGE
)
    RETURN PLS_INTEGER;
```

Parameters

Table 4–91 MSGFREE Function Parameters

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

Return Values

Table 4–92 MSGFREE Return Values

Value	Description
PLS_INTEGER	Indicates the type of the last message in the chain. The function might return any of the following values: ■ DBMS_LDAP.LDAP_RES_BIND ■ DBMS_LDAP.LDAP_RES_SEARCH_ENTRY ■ DBMS_LDAP.LDAP_RES_SEARCH_REFERENCE ■ DBMS_LDAP.LDAP_RES_SEARCH_RESULT ■ DBMS_LDAP.LDAP_RES_MODIFY ■ DBMS_LDAP.LDAP_RES_ADD ■ DBMS_LDAP.LDAP_RES_DELETE ■ DBMS_LDAP.LDAP_RES_MODDN ■ DBMS_LDAP.LDAP_RES_COMPARE ■ DBMS_LDAP.LDAP_RES_EXTENDED

Exceptions

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

Usage Notes

N / A

See Also

`DBMS_LDAP.search_s()`, `DBMS_LDAP.search_st()`.

FUNCTION ber_free

This function frees the memory associated with a handle to BER ELEMENT.

Syntax

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

Parameters

Table 4–93 BER_FREE Function Parameters

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

Return Values

N / A

Exceptions

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

Usage Notes

N / A

See Also

`DBMS_LDAP.first_attribute(),DBMS_LDAP.next_attribute().`

5

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

The standardized file format for directory entries is as follows:

```
dn: distinguished_name
attribute_type: attribute_value
.
.
.
objectClass: object_class_value
.
.
.
```

Property	Value	Description
dn:	*RDN,RDN,RDN,* ...	Separate RDNs with commas.
attribute:	*attribute_value*	This line repeats for every attribute in the entry, and for every attribute value in multi-valued attributes.
objectClass:	*object_class_ value*	This line repeats for every object class.

The following example shows a file entry for an employee. The first line contains the DN. The lines that follow the DN begin with the mnemonic for an attribute, followed by the value to be associated with that attribute. Note that each entry ends with lines defining the object classes for the entry.

```
dn: cn=Suzie Smith,ou=Server Technology,o=Acme, c=US
cn: Suzie Smith
cn: SuzieS
sn: Smith
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 NLS with LDIF files.

Command-Line Tools Syntax

This section tells you how to use the following tools:

- ldapadd Syntax
- ldapaddmt Syntax
- ldapbind Syntax
- ldapcompare Syntax
- ldapdelete Syntax
- ldapmoddn Syntax
- ldapmodify Syntax
- ldapmodifymt 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 5-15.

> **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 5-2.

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 *binddn*	When authenticating to the directory, specifies doing so as the entry specified in *binddn*. Use this with the `-w` *password* option.
-E "*character_set*"	Specifies native character set encoding. See the chapter on NLS in *Oracle Internet Directory Administrator's Guide*.
-f *filename*	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 5-2.
-h *ldaphost*	Connects to *ldaphost*, rather than to the default host, that is, your local computer. *ldaphost* can be a computer name or an IP address.
-K	Same as `-k`, but performs only the first step of the Kerberos bind

Optional Arguments	Descriptions
-k	Authenticates using Kerberos authentication instead of simple authentication. To enable this option, you must compile with KERBEROS defined. You must already have a valid ticket granting ticket.
`-M`	Instructs the tool to send the `ManageDSAIT` control to the server. The `ManageDSAIT` control instructs the server not to send referrals to clients. Instead a referral entry is returned as a regular entry.
-n	Shows what would occur without actually performing the operation
-O *ref_hop_limit*	Specifies the number of referral hops that a client should process. The default value is 5.
-p *directory_server_port_number*	Connects to the directory on TCP port *directory_server_port_number*. If you do not specify this option, the tool connects to the default port (389).
-P *wallet_password*	Specifies wallet password required for one-way or two-way SSL connections
-U *SSLAuth*	Specifies SSL authentication mode: ■ 1 for no authentication required ■ 2 for one way authentication required ■ 3 for two way authentication required
-v	Specifies verbose mode
-V *ldap_version*	Specifies the version of the LDAP protocol to use. The default value is 3, which causes the tool to use the LDAP v3 protocol. A value of 2 causes the tool to use the LDAP v2 protocol.
-w *password*	Provides the password required to connect

-W *wallet_location*	Specifies wallet location required for one-way or two-way SSL connections. For example, on Solaris, you could set this parameter as follows:
	`-W "file:/home/my_dir/my_wallet"`
	On Windows NT, you could set this parameter as follows:
	`-W "file:C:\my_dir\my_wallet"`

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 5-2.

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 *binddn*	When authenticating to the directory, specifies doing so as the entry is specified in *binddn*. Use this with the `-w password` option.
-E "*character_set*"	Specifies native character set encoding. See the chapter on NLS in *Oracle Internet Directory Administrator's Guide*.
-h *ldaphost*	Connects to *ldaphost*, rather than to the default host, that is, your local computer. *ldaphost* can be a computer name or an IP address.
-K	Same as -k, but performs only the first step of the kerberos bind

Optional Arguments	Descriptions
-k	Authenticates using Kerberos authentication instead of simple authentication. To enable this option, you must compile with KERBEROS defined. You must already have a valid ticket granting ticket.
`-M`	Instructs the tool to send the `ManageDSAIT` control to the server. The `ManageDSAIT` control instructs the server not to send referrals to clients. Instead a referral entry is returned as a regular entry.
-n	Shows what would occur without actually performing the operation.
-O *ref_hop_limit*	Specifies the number of referral hops that a client should process. The default value is 5.
-p *ldapport*	Connects to the directory on TCP port *ldapport*. If you do not specify this option, the tool connects to the default port (389).
-P *wallet_password*	Specifies wallet password required for one-way or two-way SSL connections
-T	Sets the number of threads for concurrently processing entries
-U *SSLAuth*	Specifies SSL Authentication Mode: ■ 1 for no authentication required ■ 2 for one way authentication required ■ 3 for two way authentication required
-v	Specifies verbose mode
-V *ldap_version*	Specifies the version of the LDAP protocol to use. The default value is 3, which causes the tool to use the LDAP v3 protocol. A value of 2 causes the tool to use the LDAP v2 protocol.
-w *password*	Provides the password required to connect

-W *wallet_location*	Specifies wallet location required for one-way or two-way SSL connections. For example, on Solaris, you could set this parameter as follows:
	`-W "file:/home/my_dir/my_wallet"`
	On Windows NT, you could set this parameter as follows:
	`-W "file:C:\my_dir\my_wallet"`

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 *binddn*	When authenticating to the directory, specifies doing so as the entry specified in *binddn*. Use this with the `-w` *password* option.
-E "*.character_set*"	Specifies native character set encoding. See the chapter on NLS in *Oracle Internet Directory Administrator's Guide*.
-h *ldaphost*	Connects to *ldaphost*, rather than to the default host, that is, your local computer. *ldaphost* can be a computer name or an IP address.
-n	Shows what would occur without actually performing the operation
-p *ldapport*	Connects to the directory on TCP port *ldapport*. If you do not specify this option, the tool connects to the default port (389).
-P *wallet_password*	Specifies the wallet password required for one-way or two-way SSL connections
-U *SSLAuth*	Specifies SSL authentication mode: ■ 1 for no authentication required ■ 2 for one way authentication required ■ 3 for two way authentication required
-V *ldap_version*	Specifies the version of the LDAP protocol to use. The default value is 3, which causes the tool to use the LDAP v3 protocol. A value of 2 causes the tool to use the LDAP v2 protocol.
-w *password*	Provides the password required to connect

-W *wallet_location*	Specifies wallet location required for one-way or two-way SSL connections. For example, on Solaris, you could set this parameter as follows:
	`-W "file:/home/my_dir/my_wallet"`
	On Windows NT, you could set this parameter as follows:
	`-W "file:C:\my_dir\my_wallet"`

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 *attribute name*	Specifies the attribute on which to perform the compare
-b "*basedn*"	Specifies the distinguished name of the entry on which to perform the compare
-v *attribute value*	Specifies the attribute value to compare

Optional Arguments	Descriptions
-D *binddn*	When authenticating to the directory, specifies doing so as the entry is specified in *binddn*. Use this with the `-w` *password* option.
-d *debug-level*	Sets the debugging level. See the chapter on managing a directory server in *Oracle Internet Directory Administrator's Guide*.
-E "*character_set*"	Specifies native character set encoding. See chapter on NLS in *Oracle Internet Directory Administrator's Guide*.
-f *filename*	Specifies the input filename
-h *ldaphost*	Connects to *ldaphost*, rather than to the default host, that is, your local computer. *ldaphost* can be a computer name or an IP address.
`-M`	Instructs the tool to send the `ManageDSAIT` control to the server. The `ManageDSAIT` control instructs the server not to send referrals to clients. Instead a referral entry is returned as a regular entry.
-O *ref_hop_limit*	Specifies the number of referral hops that a client should process. The default value is 5.
-p *ldapport*	Connects to the directory on TCP port *ldapport*. If you do not specify this option, the tool connects to the default port (389).

Optional Arguments	Descriptions
-P *wallet_password*	Specifies wallet password (required for one-way or two-way SSL connections)
-U *SSLAuth*	Specifies SSL authentication mode: ■ 1 for no authentication required ■ 2 for one way authentication required ■ 3 for two way authentication required
-V *ldap_version*	Specifies the version of the LDAP protocol to use. The default value is 3, which causes the tool to use the LDAP v3 protocol. A value of 2 causes the tool to use the LDAP v2 protocol.
-w *password*	Provides the password required to connect

-W *wallet_location*	Specifies wallet location required for one-way or two-way SSL connections. For example, on Solaris, you could set this parameter as follows:
	`-W "file:/home/my_dir/my_wallet"`
	On Windows NT, you could set this parameter as follows:
	`-W "file:C:\my_dir\my_wallet"`

ldapdelete Syntax

The ldapdelete command-line tool enables you to remove entire entries from the directory that you specify in the command line.

ldapdelete uses this syntax:

`ldapdelete [`*`arguments`*`] ["`*`entry_DN`*`" | -f `*`input_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
-D *binddn*	When authenticating to the directory, uses a full DN for the *binddn* parameter; typically used with the `-w` *password* option.

Optional Arguments	Descriptions
-d *debug-level*	Sets the debugging level. See the chapter on managing a directory server in *Oracle Internet Directory Administrator's Guide*.
-E "*character_set*"	Specifies native character set encoding. See the chapter on NLS in *Oracle Internet Directory Administrator's Guide*.
-f *input_filename*	Specifies the input filename
-h *ldaphost*	Connects to *ldaphost*, rather than to the default host, that is, your local computer. *ldaphost* can be a computer name or an IP address.
-k	Authenticates using authentication instead of simple authentication. To enable this option, you must compile with Kerberos defined. You must already have a valid ticket granting ticket.
`-M`	Instructs the tool to send the `ManageDSAIT` control to the server. The `ManageDSAIT` control instructs the server not to send referrals to clients. Instead a referral entry is returned as a regular entry.
-n	Shows what would be done, but doesn't actually delete
-O *ref_hop_limit*	Specifies the number of referral hops that a client should process. The default value is 5.
-p *ldapport*	Connects to the directory on TCP port *ldapport*. If you do not specify this option, the tool connects to the default port (389).
-P *wallet_password*	Specifies wallet password required for one-way or two-way SSL connections
-U *SSLAuth*	Specifies SSL authentication mode: ■ 1 for no authentication required ■ 2 for one way authentication required ■ 3 for two way authentication required
-v	Specifies verbose mode
-V *ldap_version*	Specifies the version of the LDAP protocol to use. The default value is 3, which causes the tool to use the LDAP v3 protocol. A value of 2 causes the tool to use the LDAP v2 protocol.
-w *password*	Provides the password required to connect.

Optional Arguments	Descriptions

-W *wallet_location*	Specifies wallet location required for one-way or two-way SSL connections. For example, on Solaris, you could set this parameter as follows:
	`-W "file:/home/my_dir/my_wallet"`
	On Windows NT, you could set this parameter as follows:
	`-W "file:C:\my_dir\my_wallet"`

ldapmoddn Syntax

The ldapmoddn command-line tool enables you to modify the DN or RDN of an entry.

ldapmoddn uses this syntax:

```
ldapmoddn [arguments]
```

The following example uses ldapmoddn to modify the RDN component of a DN from `"cn=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 "*basedn*"	Specifies DN of the entry to be moved

Optional Arguments	Descriptions
-D *binddn*	When authenticating to the directory, do so as the entry is specified in *binddn*. Use this with the `-w` *password* option.
-E "*character_set*"	Specifies native character set encoding. See the chapter on NLS in *Oracle Internet Directory Administrator's Guide*.
-f *filename*	Specifies the input filename
-h *ldaphost*	Connects to *ldaphost*, rather than to the default host, that is, your local computer. *ldaphost* can be a computer name or an IP address.

Optional Arguments	Descriptions
`-M`	Instructs the tool to send the `ManageDSAIT` control to the server. The `ManageDSAIT` control instructs the server not to send referrals to clients. Instead a referral entry is returned as a regular entry.
-N *newparent*	Specifies new parent of the RDN
-O *ref_hop_limit*	Specifies the number of referral hops that a client should process. The default value is 5.
-p *ldapport*	Connects to the directory on TCP port *ldapport*. If you do not specify this option, the tool connects to the default port (389).
-P *wallet_password*	Specifies wallet password required for one-way or two-way SSL connections
-r	Specifies that the old RDN is not retained as a value in the modified entry. If this argument is not included, the old RDN is retained as an attribute in the modified entry.
-R *newrdn*	Specifies new RDN
-U *SSLAuth*	Specifies SSL authentication mode: ■ 1 for no authentication required ■ 2 for one way authentication required ■ 3 for two way authentication required
-V *ldap_version*	Specifies the version of the LDAP protocol to use. The default value is 3, which causes the tool to use the LDAP v3 protocol. A value of 2 causes the tool to use the LDAP v2 protocol.
-w *password*	Provides the password required to connect.

-W *wallet_location*	Specifies wallet location required for one-way or two-way SSL connections. For example, on Solaris, you could set this parameter as follows:
	`-W "file:/home/my_dir/my_wallet"`
	On Windows NT, you could set this parameter as follows:
	`-W "file:C:\my_dir\my_wallet"`

ldapmodify Syntax

The ldapmodify tool enables you to act on attributes.

ldapmodify uses this syntax:

```
ldapmodify [arguments] -f 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 5-2.

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 *binddn*	When authenticating to the directory, specifies doing so as the entry is specified in *binddn*. Use this with the `-w` *password* option.
-E "*character_set*"	Specifies native character set encoding. See the chapter on NLS in *Oracle Internet Directory Administrator's Guide*.
-h *ldaphost*	Connects to *ldaphost*, rather than to the default host, that is, your local computer. *ldaphost* can be a computer name or an IP address.
`-M`	Instructs the tool to send the `ManageDSAIT` control to the server. The `ManageDSAIT` control instructs the server not to send referrals to clients. Instead a referral entry is returned as a regular entry.
-n	Shows what would occur without actually performing the operation.
`-o` *log_file_name*	Can be used with the `-c` option to write the erroneous LDIF entries in the logfile. You must specify the absolute path for the log file name.
-O *ref_hop_limit*	Specifies the number of referral hops that a client should process. The default value is 5.
-p *ldapport*	Connects to the directory on TCP port *ldapport*. If you do not specify this option, the tool connects to the default port (389).

Optional Arguments	Description
-P *wallet_password*	Specifies wallet password required for one-way or two-way SSL connections
-U *SSLAuth*	Specifies SSL authentication mode: ■ 1 for no authentication required ■ 2 for one way authentication required ■ 3 for two way authentication required
-v	Specifies verbose mode
-V *ldap_version*	Specifies the version of the LDAP protocol to use. The default value is 3, which causes the tool to use the LDAP v3 protocol. A value of 2 causes the tool to use the LDAP v2 protocol.
-w *password*	Overrides the default, unauthenticated, null bind. To force authentication, use this option with the `-D` option.

-W *wallet_location*	Specifies wallet location required for one-way or two-way SSL connections. For example, on Solaris, you could set this parameter as follows:
	`-W "file:/home/my_dir/my_wallet"`
	On Windows NT, you could set this parameter as follows:
	`-W "file:C:\my_dir\my_wallet"`

To run `modify`, `delete`, and `modifyrdn` operations using the `-f` flag, use LDIF for the input file format (see "LDAP Data Interchange Format (LDIF) Syntax" on page 5-2) 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 Fritchy,ou=Sales,o=Oracle,c=US
changetype:modify
add: work-phone
work-phone:510/506-7000
work-phone:510/506-7001
-
delete: home-fax
```

Unnecessary space characters in the LDIF input file, such as a space at the end of an attribute value, will cause the LDAP operations to fail.

Line 1: Every change record has, as its first line, the literal `dn:` followed by the DN value for the entry, for example:

```
dn:cn=Barbara Fritchy,ou=Sales,o=Oracle,c=US
```

Line 2: Every change record has, as its second line, the literal "`changetype:`" followed by the type of change (`add, delete, modify, modrdn`), for example:

```
changetype:modify
```

or

```
changetype:modrdn
```

Format the remainder of each record according to the following requirements for each type of change:

- `changetype:add`

  Uses LDIF format (see "LDAP Data Interchange Format (LDIF) Syntax" on page 5-2).

- `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 Fritchy,ou=Sales,o=Oracle,c=US
    changetype:modify
    add: work-phone
    work-phone:510/506-7000
    work-phone:510/506-7001
    ```

– **Delete values**. If you supply only the "delete" line, all the values for the specified attribute are deleted. Otherwise, if you specify an attribute line, you can delete specific values from the attribute:

```
delete: attribute name
[attribute name: value1]
```

For example:

```
dn:cn=Barbara Fritchy,ou=Sales,o=Oracle,c=US
changetype:modify
delete: home-fax
```

– **Replace values.** Use this option to replace all the values belonging to an attribute with the new, specified set:

```
replace:attribute name
[attribute name:value1 ...]
```

If you do not provide any attributes with "replace," then the directory adds an empty set. It then interprets the empty set as a delete request, and complies by deleting the attribute from the entry. This is useful if you want to delete attributes that may or may not exist.

For example:

```
dn:cn=Barbara Fritchy,ou=Sales,o=Oracle,c=US
changetype:modify
replace: work-phone
work-phone:510/506-7002
```

* `changetype:delete`

  This change type deletes entries. It requires no further input, since you identified the entry in Line 1 and specified a changetype of delete in Line 2.

  For example:

  ```
  dn:cn=Barbara Fritchy,ou=Sales,o=Oracle,c=US
  changetype:delete
  ```

* `changetype:modrdn`

  The line following the change type provides the new relative distinguished name using this format:

  ```
  newrdn: RDN
  ```

For example:

```
dn:cn=Barbara Fritchy,ou=Sales,o=Oracle,c=US
changetype:modrdn
newrdn: cn=Barbara Fritchy-Blomberg
```

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 5-2.

> **See Also:** "ldapmodify Syntax" on page 5-15 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 ldapadd, this flag is not required.)
-b	Specifies that you have included binary file names in the data file, which are preceded by a forward slash character.
-c	Tells ldapmodify to proceed in spite of errors. The errors will be reported. (If you do not use this option, ldapmodify stops when it encounters an error.)
-D "*binddn*"	When authenticating to the directory, specifies doing so as the entry is specified in binddn. Use this with the `-w` *password* option.
-E "*character_set*"	Specifies native character set encoding. See the chapter on NLS in *Oracle Internet Directory Administrator's Guide*.

Optional Arguments	Descriptions
-h *ldaphost*	Connects to *ldaphost*, rather than to the default host, that is, your local computer. *ldaphost* can be a computer name or an IP address.
`-M`	Instructs the tool to send the `ManageDSAIT` control to the server. The `ManageDSAIT` control instructs the server not to send referrals to clients. Instead a referral entry is returned as a regular entry.
-n	Shows what would occur without actually performing the operation.
-O *ref_hop_limit*	Specifies the number of referral hops that a client should process. The default value is 5.
-p *ldapport*	Connects to the directory on TCP port *ldapport*. If you do not specify this option, the tool connects to the default port (389).
-P *wallet_password*	Specifies wallet password required for one-way or two-way SSL connections
-T	Sets the number of threads for concurrently processing entries
-U *SSLAuth*	Specifies SSL authentication mode: ■ 1 for no authentication required ■ 2 for one way authentication required ■ 3 for two way authentication required
-v	Specifies verbose mode
-V *ldap_version*	Specifies the version of the LDAP protocol to use. The default value is 3, which causes the tool to use the LDAP v3 protocol. A value of 2 causes the tool to use the LDAP v2 protocol.
-w *password*	Overrides the default, unauthenticated, null bind. To force authentication, use this option with the -D option.

-W *wallet_location*	Specifies wallet location required for one-way or two-way SSL connections. For example, on Solaris, you could set this parameter as follows:
	`-W "file:/home/my_dir/my_wallet"`
	On Windows NT, you could set this parameter as follows:
	`-W "file:C:\my_dir\my_wallet"`

ldapsearch Syntax

The ldapsearch command-line tool enables you to search for and retrieve specific entries in the directory.

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
-b "*basedn*"	Specifies the base DN for the search
-s *scope*	Specifies search scope: base, one, or sub.

Optional Arguments	Descriptions
-A	Retrieves attribute names only (no values)
-a *deref*	Specifies alias dereferencing: never, always, search, or find
-B	Allows printing of non-ASCII values
-D *binddn*	When authenticating to the directory, specifies doing so as the entry specified in *binddn*. Use this with the `-w` *password* option.
-d *debug level*	Sets debugging level to the level specified (see the chapter on managing a directory server in *Oracle Internet Directory Administrator's Guide*.)
-E "*character_set*"	Specifies native character set encoding. See the chapter on NLS in *Oracle Internet Directory Administrator's Guide*.
-f *file*	Performs sequence of searches listed in *file*
-F sep	Prints '`sep`' instead of '`=`' between attribute names and values
-h *ldaphost*	Connects to *ldaphost*, rather than to the default host, that is, your local computer. *ldaphost* can be a computer name or an IP address.
-L	Prints entries in LDIF format (`-B` is implied)

Optional Arguments	Descriptions
-l *timelimit*	Specifies maximum time (in seconds) to wait for ldapsearch command to complete
`-M`	Instructs the tool to send the `ManageDSAIT` control to the server. The `ManageDSAIT` control instructs the server not to send referrals to clients. Instead a referral entry is returned as a regular entry.
-n	Shows what would be done without actually searching
-O *ref_hop_limit*	Specifies the number of referral hops that a client should process. The default value is 5.
-p *ldapport*	Connects to the directory on TCP port *ldapport*. If you do not specify this option, the tool connects to the default port (389).
-P *wallet_password*	Specifies wallet password (required for one-way or two-way SSL connections)
-S *attr*	Sorts the results by attribute *attr*
-t	Writes to files in `/tmp`
-u	Includes user friendly entry names in the output
-U *SSLAuth*	Specifies the SSL authentication mode: ■ 1 for no authentication required ■ 2 for one way authentication required ■ 3 for two way authentication required
-v	Specifies verbose mode
-V *ldap_version*	Specifies the version of the LDAP protocol to use. The default value is 3, which causes the tool to use the LDAP v3 protocol. A value of 2 causes the tool to use the LDAP v2 protocol.
-w *passwd*	Specifies bind passwd for simple authentication

-W *wallet_location*	Specifies wallet location required for one-way or two-way SSL connections. For example, on Solaris, you could set this parameter as follows:
	`-W "file:/home/my_dir/my_wallet"`
	On Windows NT, you could set this parameter as follows:
	`-W "file:C:\my_dir\my_wallet"`

-z *sizelimit*	Specifies maximum number of entries to retrieve

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"
```

Catalog Management Tool Syntax

Oracle Internet Directory uses indexes to make attributes available for searches. When Oracle Internet Directory is installed, the entry `cn=catalogs` lists available attributes that can be used in a search. Only those attributes that have an equality matching rule can be indexed.

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.

Before running the Catalog Management tool, unset the LANG variable. After you finish running Catalog Management tool, set the LANG variable back to its original value.

To unset LANG:

- Using Korn shell:

  ```
  UNSET LANG
  ```

- Using C shell:

  ```
  UNSETENV LANG
  ```

The Catalog Management tool uses this syntax:

`catalog.sh -connect` *`net_service_name`* `{add|delete} {-attr` *`attr_name`*`|-file` *`filename}`*

Mandatory Argument	Description
- connect *net_service_name*	Specifies the net service name to connect to the directory database **See Also:** *Oracle Net Services Administrator's Guide*

Optional Arguments	Descriptions
- add -attr *attr_name*	Indexes the specified attribute
- delete -attr *attr_name*	Drops the index from the specified attribute
- add -file *filename*	Indexes attributes (one per line) in the specified file
-delete -file *filename*	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
```

After you finish running the Catalog Management tool, set the LANG variable back to its original value.

To set LANG:

- Using Korn shell:

  ```
  SET LANG=appropriate_language; EXPORT LANG
  ```

- Using C shell:

  ```
  SETENV LANG appropriate_language
  ```

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

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 Oracle9*i* release release 9.0.1 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, any directory or repository other than the **central directory**. In such an environment, Oracle Internet Directory serves as the central directory, and all other directories are connected directories. Synchronization always happens between Oracle Internet Directory and a connected directory.

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.

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 contains configuration information required for synchronization.

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 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 system agent (DSA)**. 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.

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 **Oracle directory integration server (DIS)**

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)**

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 the 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 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**.

import file

See **import data file**.

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 **Oracle directory integration server (DIS)**.

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 reference**s (also called referrals) to subordinate naming contexts. It can range in size from a single entry to the entire DIT.

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.

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 the 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 allows 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 allows various information repositories to synchronize with Oracle Internet Directory and to form a single virtual directory.

Oracle directory integration server (DIS)

In an Oracle Directory Integration platform environment, the server that drives the synchronization of data between Oracle Internet Directory and a **connected directory**.

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 Oracle9*i*.

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.

Oracle9*i* Replication

A feature in Oracle9*i* that allows 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.

partner agent

A directory integration agent for which the Oracle Directory Integration server performs mapping, scheduling, and error handling.

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.

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, but does so as a proxy user. 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

See **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`.

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 Sockets 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 the above 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 Sockets Layer (SSL)**.

subclass

An object class derived from another object class. The object class from which it is derived is called its **superclass**.

subschema DN

The list of DIT areas having independent schema definitions.

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 point**s
- 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.

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.

subACLSubentry

A specific type of subentry that contains ACL information.

subSchemaSubentry

A specific type of **subentry** containing schema information.

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."

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 per 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 allows 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**.

UCS-2

Fixed-width 16-bit **Unicode**. Each character occupies 16 bits of storage. The Latin-1 characters are the first 256 code points in this standard, so it can be viewed as a 16-bit extension of Latin-1.

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 encoding of **UCS-2** which uses sequences of 1, 2, or 3 bytes per character. Characters from 0-127 (the 7-bit ASCII characters) are encoded with one byte, characters from 128-2047 require two bytes, and characters from 2048-65535 require three bytes. The Oracle character set name for this is UTF-8 (for the Unicode 2.1 standard). The standard has left room for expansion to support the UCS4 characters with sequences of 4, 5, and 6 bytes per character.

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

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

W