Oracle® Internet Directory

Application Developer's Guide

Release 2.1.1

September 2000

Part No. A86082-01

ORACLE®

Oracle Internet Directory Application Developer's Guide, Release 2.1.1

Part No. A86082-01

Primary Author: Richard Smith

Contributors: Saheli Dey, Rajinder Gupta, Ashish Kolli Stephen Lee, David Lin, Radikha Moolky, David Saslav

Contents

3 PL/SQL API

4 Command Line Tools Syntax

Index

Send Us Your Comments

Oracle Internet Directory Application Developer's Guide, Release 2.1.1

Part No. A86082-01

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:

- E-mail - infodev@us.oracle.com
- FAX - (650) 506-7228. Attn: Oracle Internet Directory Documentation Manager
- Postal service:
  Oracle Corporation
  Oracle Internet Directory Documentation Manager
  500 Oracle Parkway, 4op7
  Redwood Shores, CA 94065
  U.S.A.

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.

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 2.1.1. It also lists the other components of Oracle Internet Directory and the platforms it supports.
Chapter 2, "C API"	Introduces the Oracle Internet Directory C API and provides examples of how to use it
Chapter 3, "PL/SQL API"	Introduces the PL/SQL API contained in a PL/SQL package called DBMS_LDAP. It also contains examples of how to use it.

Chapter 4, "Command Line Tools Syntax"	Provides syntax, usage notes, and examples for using LDAP Data Interchange Format (LDIF) and LDAP command line tools

Related Documentation

Oracle Internet Directory Administrator's Guide.

Oracle8*i* documentation set

Chadwick, David. *Understanding X.500 The Directory.* Thomson Computer Press, 1996. This book is now out of print, but is available online at: http://www.salford.ac.uk/its024/Version.Web/Contents.htm

Hodges, Jeff, Staff Scientist, Oblix, Inc., http://www.kingsmountain.com/ldapRoadmap.shtml

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.

Kosiur, Dave, LDAP: "The next-generation directory?," *SunWorld Online*, October 1997.

Radicati, Sara, *X.500 Directory Services, Technology and Deployment*, International Thomson Computer Press, 1994.

University of Michigan LDAP Repository, http://www.umich.edu/~dirsvcs/ldap/index.html

Conventions

The following conventions are used in this manual:

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
. . .	Horizontal ellipsis points in statements or commands mean that parts of the statement or command have been omitted.

Convention	Meaning
bold	Boldface text indicates text you must type in a command, or a subheading.
italics	Italics indicate: ■ In a code example, a variable for which you must supply a value ■ In regular text, special emphasis ■ Book titles
`courier`	Courier is used for user input and code examples.
syntax	This typeface is used for syntax explanations in code examples.
< >	In code examples, angle brackets may enclose user-supplied names.
[ ]	Brackets enclose a choice of optional items from which you can choose one or none.
{ }	Braces enclose a choice of required items from which you can choose one.

1

Introduction

This chapter briefly describes the intended audience and components of Oracle Internet Directory Software Developer's Kit Release 2.1.1. 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 2.1.1

Oracle Internet Directory SDK Release 2.1.1 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 2.1.1 consists of:

- An LDAP Version 2-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 2.1.1, 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*
- *Oracle Internet Directory Installation Guide*

Operating Systems Supported

Oracle Internet Directory, both servers and clients, support the following 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

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 Release 2.1.1 is based on:

- LDAP Version 2 C API
- Oracle extensions to support SSL

Oracle Internet Directory SDK C API Release 2.1.1 supports the following modes:

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

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 2-7 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:

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

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 2-7

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 2.1.1 of the API supports only Oracle Wallet. You can create wallets using Oracle Wallet Manager.

Summary of LDAP C API

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

This section contains these topics:

- Initializing and Ending LDAP Sessions
- Authenticating to an LDAP Server
- Performing LDAP Operations
- Getting Search Results
- Working with Distinguished Names
- Handling Errors
- Freeing Memory

Initializing and Ending LDAP Sessions

ldap_open()	Open a connection to an LDAP server
ldap_unbind()	End an LDAP session

Authenticating to an LDAP Server

ldap_ bind()	General authentication to an LDAP server
ldap_bind_s()	
ldap_simple_bind()	Simple authentication to an LDAP server
ldap_simple_bind_s()	

Performing LDAP Operations

ldap_add() / ldap_add_s()	Add a new entry to the directory
ldap_modify() / ldap_modify_s()	Modify an entry in the directory

ldap_delete() / ldap_delete_s()	Delete an entry from the directory
ldap_modrdn() / ldap_modrdn_s()	Modify the RDN of an entry in the directory
ldap_search() / ldap_search_s()	Search the directory
ldap_search_st()	Search the directory with a timeout value
ldap_compare() / ldap_compare_s()	Compare entries in the directory
ldap_result()	Check the results of an asynchronous operation
ldap_abandon()	Cancel an asynchronous operation

Getting Search Results

ldap_get_dn()	Get the distinguished name for an entry
ldap_first_entry()	Get the first entry in a chain of search results
ldap_next_entry()	Get the next entry in a chain of search results
ldap_count_ entries()	Count the number of entries in a chain of search results
ldap_first_ attribute()	Get the name of the first attribute in an entry
ldap_next_ attribute()	Get the name of the next attribute in an entry
ldap_get_values()	Get the string values of an attribute
ldap_get_values_ len()	Get the binary values of an attribute
ldap_count_ values()	Count the string values of an attribute
ldap_count_ values_len()	Count the binary values of an attribute

Working with Distinguished Names

ldap_get_dn()	Get the distinguished name for an entry
ldap_explode_dn()	Split up a distinguished name into its components
ldap_dn2ufn()	Converts the name into a more user friendly format

Handling Errors

ldap_result2error()	Returns the error code from result message.
ldap_err2string()	Get the error message for a specific error code
ldap_perror	Prints the message supplied in message.

Freeing Memory

ldap_memfree()	Free memory allocated by an LDAP API function call
ldap_msgfree()	Free the memory allocated for search results or other LDAP operation results
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
ber_free()	Free the memory allocated for a BerElement structure

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 usage 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 files located at $ORACLE_HOME/ldap/public/ldap.h.
- The libraries located at $ORACLE_HOME/lib/libldapclnt8.a

Building a Sample Search Tool

The Oracle Internet Directory SDK Release 2.1.1 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 ldapsearch 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 server or a third party LDAP server.

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 Oracle 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).

3

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 Oracle Internet Directory PL/SQL API
- Sample PL/SQL Usage
- Building Applications with PL/SQL LDAP API
- Dependencies and Limitations
- PL/SQL Reference

About the Oracle Internet Directory PL/SQL API

The Oracle Internet Directory 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. 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.

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:

- It cannot be used in the multi-threaded server (MTS) mode of the database.
- 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 3–1 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 3–1 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 3–1 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.

Exception Summary

The DBMS_LDAP package shipped with RDBMS 8.1.7 can generate the following exceptions

Table 3–2 DBMS_LDAP Exception Summary

Exception Name	Oracle Error Number	Cause of Exception
general_error	31202	Raised anytime an error is encountered that does not have a specific PL / SQL exception associated with it. The error string contains the description of the problem in the local language of the user.
init_failed	31203	Raised by DBMS_LDAP.init() if there are some problems.
invalid_session	31204	Raised by all functions and procedures in the DBMS_LDAP package if they are passed an invalid session handle.
invalid_auth_method	31205	Raised by DBMS_LDAP.bind_s() if the authentication method requested is not supported.
invalid_search_scope	31206	Raised by all of the 'search' functions if the scope of the search is invalid.
invalid_search_time_val	31207	Raised by time based search function: DBMS_ LDAP.search_st() if it is given an invalid value for the time limit.
invalid_message	31208	Raised by all functions that iterate through a result-set for getting entries from a search operation if the message handle given to them is invalid.
count_entry_error	31209	Raised by DBMS_LDAP.count_entries if it cannot count the entries in a given result set.
get_dn_error	31210	Raised by DBMS_LDAP.get_dn if the DN of the entry it is retrieving is NULL.
invalid_entry_dn	31211	Raised by all the functions that modify / add / rename an entry if they are presented with an invalid entry DN.
invalid_mod_array	31212	Raised by all functions that take a modification array as an argument if they are given an invalid modification array.

Table 3–2 DBMS_LDAP Exception Summary

Exception Name	Oracle Error Number	Cause of Exception
invalid_mod_option	31213	Raised by DBMS_LDAP.populate_mod_array if the modification option given is anything other than MOD_ADD, MOD_DELETE or MOD_ REPLACE.
invalid_mod_type	31214	Raised by DBMS_LDAP.populate_mod_array if the attribute type that is being modified is NULL.
invalid_mod_value	31215	Raised by DBMS_LDAP.populate_mod_array if the modification value parameter for a given attribute is NULL.
invalid_rdn	31216	Raised by all functions and procedures that expect a valid RDN if the value of the RDN is NULL.
invalid_newparent	31217	Raised by DBMS_LDAP.rename_s if the new parent of an entry being renamed is NULL.
invalid_deleteoldrdn	31218	Raised by DBMS_LDAP.rename_s if the deleteoldrdn parameter is invalid.
invalid_notypes	31219	Raised by DBMS_LDAP.explode_dn if the notypes parameter is invalid.
invalid_ssl_wallet_loc	31220	Raised by DBMS_LDAP.open_ssl if the wallet location is NULL but the SSL authentication mode requires a valid wallet.
invalid_ssl_wallet_password	31221	Raised by DBMS_LDAP.open_ssl if the wallet password given is NULL.
invalid_ssl_auth_mode	31222	Raised by DBMS_LDAP.open_ssl if the SSL authentication mode is not one of 1, 2 or 3.
mts_mode_not_supported	31398	Raised by the functions init(), bind_s() or simple_ bind_s() if they are ever invoked in MTS mode.

Data-Type Summary

The DBMS_LDAP package uses the following data-types.

Table 3–3 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 3–4 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 3–5 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 3–6 INIT Function Exceptions

Exception	Description
init_failed	Raised when there is a problem contacting the LDAP server.
mts_mode_not_supported	Raised if DBMS_LDAP.init() is invoked from a user session that is logged onto the database using an MTS service.
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 3–7 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 3–8 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 3–9 SIMPLE_BIND_S Function Exceptions

Exception	Description
invalid_session	Raised if the session handle 'ld' is invalid.
mts_mode_not_supported	Raised if DBMS_LDAP.init() is invoked from a user session that is logged onto as an MTS service.
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 3–10 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 3–11 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 3–12 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.
mts_mode_not_supported	Raised if invoked from a user session that is logged onto an MTS service.
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 3–13 UNBIND_S Function Parameters

Parameter	Description
ld	A valid LDAP session handle.

Return Values

Table 3–14 UNBIND_S Function Return Values

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

Exceptions

Table 3–15 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 3–16 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 3–17 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 3–18 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 3–19 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.

Table 3–19 SEARCH_S 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.
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 3–20 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 3–21 SEARCH_S Function Exceptions

Exception	Description
invalid_session	Raised if the session handle 'ld' is invalid.
invalid_search_scope	Raised i 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 3–22 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 3–22 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 3–23 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 3–24 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 3–25 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 3–26 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 3–27 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 3–28 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 3–29 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 3–30 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 3–31 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 3–32 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 3–33 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 3–34 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 3–35 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 3–36 FIRST_ATTRIBUTE Function Exceptions

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

Usage Notes

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

See Also

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

FUNCTION next_attribute

The function next_attribute() fetches the next attribute of a given entry in the result set.

Syntax

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

Parameters

Table 3–37 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 3–38 NEXT_ATTRIBUTE Function Return Values

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

Exceptions

Table 3–39 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 3–40 GET_DN Function Parameters

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

Return Values

Table 3–41 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 3–42 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 3–43 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 3–44 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 3–45 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 3–46 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 3–47 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 3–48 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 3–49 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 3–50 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 3–51 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 3–52 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 3–53 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 3–54 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 3–55 ERR2STRING Function Parameters

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

Return Values

Table 3–56 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 3–57 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 3–58 CREATE_MOD_ARRAY Function Parameters

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

Return Values

Table 3–59 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 3–60 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 3–61 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 3–62 POPULATE_MOD_ARRAY (String Version) Procedure Return Values

Value	Description
N / A	

Exceptions

Table 3–63 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 3–64 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 3–65 POPULATE_MOD_ARRAY (Binary Version) Procedure Return Values

Value	Description
N/A	

Exceptions

Table 3–66 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 3–67 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 3–68 MODIFY_S Function Return Values

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

Exceptions

Table 3–69 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 3–70 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 3–71 ADD_S Function Return Values

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

Exceptions

Table 3–72 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 3–73 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 3–74 FREE_MOD_ARRAY Procedure Return Value

Value	Description
N/A	

Exceptions

Table 3–75 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 3–76 COUNT_VALUES Function Parameters

Parameter	Description
values	The collection of string values.

Return Values

Table 3–77 COUNT_VALUES Function Return Values

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

Exceptions

Table 3–78 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 3–79 COUNT_VALUES_LEN Function Parameters

Parameter	Description
values	The collection of binary values.

Return Values

Table 3–80 COUNT_VALUES_LEN Function Return Values

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

Exceptions

Table 3–81 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,
    ewparent    IN VARCHAR2,
    deleteoldrdn IN PLS_INTEGER,
    serverctrls  IN LDAPCONTROL,
    clientctrls  IN LDAPCONTROL
)
    RETURN PLS_INTEGER;
```

Parameters

Table 3–82 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 3–83 RENAME_S Function Return Values

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

Exceptions

Table 3–84 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 3–85 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 3–86 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 3–87 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 3–88 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 3–89 OPEN_SSL Function Return Values

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

Exceptions

Table 3–90 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()`.

4

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 4-13.

> **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 4-2.

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 4-2.
-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.
-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 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
-w *password*	Provides the password required to connect
-W *wallet_location*	Specifies wallet location required for one-way or two-way SSL connections

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 4-2.

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
-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.
-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 wallet password required for one-way or two-way SSL connections
-T	Sets the number of threads for concurrently processing entries

Optional Arguments	Descriptions
-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
-w *password*	Provides the password required to connect
-W *wallet_location*	Specifies wallet location required for one-way or two-way SSL connections

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

Optional Arguments	Descriptions
-w *password*	Provides the password required to connect
-W *wallet_location*	Specifies wallet location (required for one-way or two-way SSL connections)

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.
-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
-w *password*	Provides the password required to connect
-W *wallet_location*	Specifies wallet location required for one-way or two-way SSL connections

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"
```

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.
-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 *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.

Optional Arguments	Descriptions
-n	Shows what would be done, but doesn't actually delete
-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
-w *password*	Provides the password required to connect.
-W *wallet_location*	Specifies wallet location required for one-way or two-way SSL connections

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.
-N *newparent*	Specifies new parent of the RDN
-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
-w *password*	Provides the password required to connect.
-W *wallet_location*	Specifies wallet location required for one-way or two-way SSL connections

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 4-2.

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.
-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 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

Optional Arguments	Description
-w *password*	Overrides the default, unauthenticated, null bind. To force authentication, use this option with the `-D` option.
-W *wallet_location*	Specifies wallet location (required for one-way or two-way SSL connections)

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 4-2) with the specifications noted below:

Always separate entries with a blank line.

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 4-2).

- `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:delete
delete: home-fax
```

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

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

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

For example:

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

* `changetype:delete`

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

  For example:

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

* `changetype:modrdn`

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

  `newrdn: RDN`

  For example:

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

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 4-2.

> **See Also:** "ldapmodify Syntax" on page 4-13 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*.
-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 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
-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

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. For further information about this standard, search for the standard at: http://www.ietf.org/rfc/rfc2254.txt

Separate attributes with a space. If you do not list any attributes, all attributes are retrieved.

Mandatory Arguments	Descriptions
-b *basedn*	Specifies base dn for 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)
-l *timelimit*	Specifies maximum time (in seconds) to wait for ldapsearch command to complete

Optional Arguments	Descriptions
-n	Shows what would be done without actually searching
-p *ldapport*	Connects to the directory on TCP port *ldapport*. If you do not specify this option, the tool connects to the default port (389).
-P *wallet_password*	Specifies wallet password (required for one-way or two-way SSL connections)
-S *attr*	Sorts the results by attribute *attr*
-t	Writes to files in `/tmp`
-u	Includes user friendly entry names in the output
-U *SSLAuth*	Specifies the SSL authentication mode: ■ 1 for no authentication required ■ 2 for one way authentication required ■ 3 for two way authentication required
-v	Specifies verbose mode
-w *passwd*	Specifies bind passwd for simple authentication
-W *wallet_location*	Specifies wallet location required for one-way or two-way SSL connections
-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: Sub-Tree 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 and Attribute Options 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
```

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 fails:

```
ldapsearch -p 389 -h myhost -b "c=us" -s sub "cn;lang-it=Giovanni
```

Example 6: 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 2000 and 06 April 2000:

```
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:** *Net8 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

Index

A

B

C

M

O

P

R

S

T

W