z/OS

SecureWay® Security Server LDAP Client Programming

SC24-5924-00

z/OS

SecureWay® Security Server LDAP Client Programming

SC24-5924-00

Note

Before using this information and the product it supports, be sure to read the general information under Appendix D, ″Notices.″

Acknowledgements

Some of the material contained in this document is a derivative of LDAP documentation provided with the University of Michigan LDAP reference implementation (Version 3.3). Copyright ©1992-1996, Regents of the University of Michigan, All Rights Reserved.

This product includes software developed by the University of California, Berkeley and its contributors.

This product includes software developed by NEC Systems Laboratory.

First Edition (March 2001)

This edition, SC24-5924-00, applies to Version 1 Release 1 of z/OS SecureWay Security Server (program number 5694-A01), and to all subsequent releases until otherwise indicated in new editions.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

Comments may be addressed to IBM at the following address:

International Business Machines Corporation
Information Development, Dept. G60
1701 North Street
Endicott, NY 13760-5553
United States of America

FAX (United States & Canada): 1+607+752-2327
FAX (Other Countries):
Your International Access Code +1+607+752-2327

IBMLink (United States customers only): GDLVME(PUBRCF)
Internet e-mail: pubrcf@vnet.ibm.com
World Wide Web: `http://www.ibm.com/servers/eserver/zseries/zos/`

If you would like a reply, be sure to include your name, address, telephone number, or FAX number. Make sure to include the following in your comment or note:

- Title and order number of this book
- Page number or topic related to your comment

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Tables

Preface

This book describes the Lightweight Directory Access Protocol (LDAP) client application development for z/OS SecureWay Security Server.

Who Should Use This Book

This document is intended for application programmers. Application programmers should be experienced and have previous knowledge of directory services.

How This Book is Organized

This book is organized in the following manner:

- "Chapter 1. LDAP Programming" on page 1 describes how to use the LDAP client application programming interface.
- "Chapter 2. LDAP Routines" on page 17 describes each LDAP client routine.
- "Chapter 3. LDAP Operation Utilities" on page 85 describes the LDAP operation utilities and how to run them.
- "Appendix A. LDAP Header Files" on page 107 shows each LDAP header file.
- "Appendix B. Sample Makefile" on page 119 shows a sample Makefile.
- "Appendix C. Example Programs" on page 121 shows examples of how to use the LDAP programming interface.

Conventions Used in This Book

This book uses the following typographic conventions:

Bold **Bold** words or characters represent API names, attributes, status codes, environment variables, parameter values, and system elements that you must enter into the system literally, such commands, options, or path names.

Italic *Italic* words or characters represent values for variables that you must supply.

`Example Font`
Examples and information displayed by the system appear in `constant width type style`.

[ ] Brackets enclose optional items in format and syntax descriptions.

{ } Braces enclose a list from which you must choose an item in format and syntax descriptions.

| A vertical bar separates items in a list of choices.

< > Angle brackets enclose the name of a key on the keyboard.

... Horizontal ellipsis points indicate that you may repeat the preceding item one or more times.

**** A backslash is used as a continuation character when entering commands from the shell that exceed one line (255 characters). If the command exceeds one line, use the backslash character \ as the last nonblank character on the line to be continued, and continue the command on the next line.

Where to Find More Information

Where necessary, this book references information in other books. For complete titles and order numbers of the books for all products that are part of z/OS, refer to *z/OS: Information Roadmap*, SA22-7500.

For a list of titles and order numbers of the books that are useful for z/OS LDAP, see “Bibliography” on page 147 .

Softcopy Publications

The z/OS Security Server library is available on a CD-ROM, *z/OS: Collection*, SK3T-4269. The CD-ROM online library collection is a set of unlicensed books for z/OS and related products that includes the IBM LIbrary Reader. This is a program that enables you to view the BookManager files. This CD-ROM also contains the Portable Document Format (PDF) files. You can view or print these files with the Adobe Acrobat reader.

Internet Sources

- **z/OS Online Library**

  The softcopy z/OS publications are also available for web browsing and for viewing or printing PDFs using the following URL:

  `http://www.ibm.com/servers/eserver/zseries/zos/bkserv`

  You can also provide comments about this book and any other z/OS documentation by visiting that URL. Your feedback is important in helping to provide the most accurate and high-quality information.

- **Accessing Licensed Books on the Web**

  z/OS licensed documentation in PDF format is available on the Internet at the IBM Resource Link Web site:

  `http://www.ibm.com/servers/resourcelink`

  Licensed books are available only to customers with a z/OS license.Access to these books requires an IBM Resource Link user ID, password, and z/OS licensed book key code. The z/OS order you received provides a memo that includes your key code.

  To obtain your IBM Resource Link user ID and password, logon to:

  `http://www.ibm.com/servers/resourcelink`

  To register for access to the z/OS licensed books:
  1. Logon to Resource Link using your Resource Link user ID and password.
  2. Select **User Profiles** located on the left-hand navigation bar.
  3. Select **Access Profile**.
  4. Select **Request Access to Licensed books**.
  5. Supply your key code where requested and select the **Submit** button.

  If you supplied the correct key code you will receive confirmation that your request is being processed.

  After your request is processed you will receive an e-mail confirmation.

  Note: You cannot access the z/OS licensed books unless you have registered for access to them and received an e-mail confirmation informing you that your request has been processed.

  To access the licensed books:
  1. Logon to Resource Link using your Resource Link user ID and password.
  2. Select **Library**.
  3. Select **zSeries**.
  4. Select **Software**.
  5. Select **z/OS**.
  6. Access the licensed book by selecting the appropriate element.

Using LookAt to Look Up Message Explanations

LookAt is an online facility that allows you to look up explanations for z/OS messages. You can also use LookAt to look up explanations of system abends.

Using LookAt to find information is faster than a conventional search because LookAt goes directly to the explanation.

LookAt can be accessed from the Internet or from a TSO command line. You can use LookAt on the Internet at:

`http://www.ibm.com/servers/eserver/zseries/zos/bkserv/lookat/lookat.html`

To use LookAt as a TSO command, LookAt must be installed on your host system. You can obtain the LookAt code for TSO from the LookAt Web site by clicking on the **News and Help** link or from the *z/OS: Collection*, SK3T-4269.

To find a message explanation from a TSO command line, simply enter: **lookat** *message-id* as in the following:

```
lookat iec192i
```

This results in direct access to the message explanation for message IEC192I.

To find a message explanation from the LookAt Web site, simply enter the message ID and select the release you are working with.

Note: Some messages have information in more than one book. For example, IEC192I has routing and descriptor codes listed in *z/OS: MVS Routing and Descriptor Codes*, SA22-7624. For such messages, LookAt prompts you to choose which book to open.

How to Send Your Comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this book or any other z/OS documentation:

- Visit the home page at: `http://www.ibm.com/servers/eserver/zseries/zos/`
- Fill out one of the forms at the back of the book and return it by mail, by fax, or by giving it to an IBM representative.

Chapter 1. LDAP Programming

The Lightweight Directory Access protocol (LDAP) was defined in response to many complaints about the complexity of interacting with an X.500 Directory Service using the full Directory Access Protocol (DAP). A number of programmers at the University of Michigan proposed and implemented a lightweight version of a directory access protocol. This work has grown into what is termed the LDAP protocol.

The LDAP support in z/OS is for client access to Directory Services that accept the LDAP protocol. The LDAP client allows programs running on z/OS UNIX to enter and extract information into and from a Directory Service. The LDAP Server, a component of the SecureWay Security Server for z/OS, can be used to store and extract information on z/OS using the LDAP protocol. See *z/OS: SecureWay Security Server LDAP Server Administration and Use* for more information.

Regarding security, two authentication methods are supported: simple authentication and certificate authentication. With simple authentication, a user ID and password are sent (in the clear) from the client to the server in order to establish who is contacting the LDAP server for information.

However, Secure Socket Layer (SSL) can be used to secure the socket connection between the client and the server. SSL can be used to encrypt the user ID and password.

With authentication:certificate;certificate authenticationcertificate authentication, the identity from the client certificate sent to the LDAP server on an SSL socket connection is used to establish who is contacting the LDAP server for information. Certificate authentication is also referred to as *SASL external bind* and is provided by the **ldap_sasl_bind** API.

This chapter focuses on the following topics:
- Defining the LDAP protocol
- The LDAP Data model, including the format of distinguished names in LDAP
- An overview of the functions supported by the LDAP client API on z/OS
- Details on compiling and link-editing a program that uses the LDAP client API
- Information on how to use the LDAP client APIs
- An example program which shows how the LDAP client API could be used as a Directory Service
- The LDAP Version 3 Client for Java

How LDAP Is Defined

The LDAP protocol is defined by a number of Internet Engineering Task Force (IETF) request for comments (RFCs). This protocol is defined in IETF RFC 1777 *Lightweight Directory Access Protocol*. Other RFCs of interest include:
- RFC 1778 - *The String Representation of Standard Attribute Syntaxes*
- RFC 1779 - *A String Representation of Distinguished Names*
- RFC 1960 - *A String Representation of LDAP Search Filters*
- RFC 1823 - *The LDAP Application Program Interface*
- RFC 2255 - *The LDAP URL Format*
- RFC 1738 - *Uniform Resource Locators (URL)*

IETF RFC 1823 defines a programming interface for using the LDAP protocol to communicate with a Directory Service that accepts the LDAP protocol. The programming interface available on z/OS is very similar to the interface defined by RFC 1823.

IETF RFC 2251 *Lightweight Directory Access Protocol (v3)* defines the so-called LDAP Version 3 specification. LDAP Version 3 is what is implemented by the LDAP client interfaces for z/OS.

The LDAP protocol is defined using ASN.1 notation. The wire protocol is defined as the Basic Encoding Rules (BER) encodings of the ASN.1-defined structures. Furthermore, these BER encoded messages are defined to be carried over a TCP/IP socket connection to a server that accepts the LDAP protocol.

Data Model

The LDAP data model is closely aligned with the X.500 data model. In this model, a Directory Service provides a hierarchically organized set of *entries*. Each of these entries is represented by an *object class* (or set of object classes). The object class of the entry determines the set of *attributes* which are required to be present in the entry as well as the set of attributes that can optionally appear in the entry. An attribute is represented by an *attribute type* and one or more *attribute values*. In addition to the attribute type and values, each attribute has an associated *syntax* which describes the type of the attribute values. Examples of attribute syntaxes include **PrintableString** and **OctetString**.

To summarize, the directory is made up of entries. Each entry contains a set of attributes. These attributes can be single or multi-valued (have one or more values associated with them). The object class of an entry determines the set of attributes that must and the set of attributes that may exist in the entry. Refer to *z/OS: DCE Application Development Guide: Directory Services*, SC24-5906 for more about the X.500 directory information model.

In XDS/XOM, a complex set of arrays of structures is used to represent a directory entry. In LDAP, this is somewhat simplified. With the LDAP API, a set of C language utility routines is used to extract attribute type and value information from directory entry information returned from an LDAP search operation. Unlike XDS/XOM, attribute values are provided to the calling program in either null-terminated character string form or in a simple structure that specifies a pointer and a length value. Further, attribute types are provided to the program as null-terminated character strings instead of object identifiers.

LDAP Names

The LDAP protocol (and API) uses so-called “typed” names to identify directory entries. In contrast, DCE CDS and the Domain Name Service (DNS) use “untyped” names to identify entries. Each directory entry is identifiable by its fully distinguished name. The distinguished name (DN) is constructed by catenating the relative distinguished names (RDNs) of each entry in the directory hierarchy leading from the root of the namespace to the entry itself. This is identical to the X.500 naming model. With LDAP, however, a distinguished name is specified using a null-terminated character string instead of a complex set of nested arrays of XOM structures. Note that an RDN can consist of multiple attribute type/value pairs.

Examples of LDAP RDNs include:

```
c=US
o=Acme International
ou=Marketing+l=Virginia
cn=Jane Doe
```

The same set of RDNs specified in the string format of X.500 names in DCE would appear as:

```
"c=US", "o=Acme International", "ou=Marketing;l=Virginia", and "cn=Jane Doe"
```

If each of these RDNs represented directory entries that appeared below the entry before it, the DN for the lowest entry in the directory (using the DCE X.500 string form) would be:

```
/c=US/o="Acme International"/ou=Marketing;l=Virginia/cn="Jane Doe"
```

The LDAP format for this DN is a bit different:

```
cn=Jane Doe, ou=Marketing+l=Virginia, o=Acme International, c=US
```

An LDAP DN is specified as a null-terminated character string in a right-to-left fashion (right-to-left refers to the ordering of RDNs from highest to lowest in the directory hierarchy). Note that embedded spaces are

taken as part of the attribute value for RDNs and do not require quotation marks. Also note that RDNs are separated by commas (,) and attribute type/value pairs within an RDN are separated by plus (+) signs. (Refer to IETF RFC 1779 for more information.)

Function Overview

The LDAP client API is provided in a set of C/C++ DLLs which is loaded at run time by applications that use the LDAP API. The DLL that externalizes the LDAP programming interfaces is called **GLDCLDAP**. The **GLDCLDAP** DLL makes use of two additional DLLs, **GLDCMMN** and **GLDSCKS**, which are loaded automatically by the **GLDCLDAP** DLL. Refer to "Compiling, Linking, and Running a Program" on page 6 for details on how to link-edit a program to use the proper form of the LDAP DLLs.

The PDS versions of the DLLs are installed into **LPALIB**. The HFS version of **GLDCLDAP** is installed to **/usr/lpp/ldapclient/lib**. Symbolic links are set at installation to this file system from **/usr/lib**.

The LDAP API consists of 53 C language functions. All function names begin with the prefix **ldap_**. The functions can be broken down into six categories as shown in Table 1.

For detailed information about each LDAP API see "Chapter 2. LDAP Routines" on page 17.

Table 1. LDAP API Functions

Category	Function name
Initialization / Termination	**ldap_init** **ldap_open** **ldap_ssl_init**, **ldap_ssl_client_init**, **ldap_ssl_start** **ldap_unbind**, **ldap_unbind_s**
Primitive Operations	**ldap_abandon** **ldap_add**, **ldap_add_s** **ldap_add_ext**, **ldap_add_ext_s** **ldap_bind**, **ldap_bind_s** **ldap_compare**, **ldap_compare_s** **ldap_compare_ext**, **ldap_compare_ext_s** **ldap_delete**, **ldap_delete_s** **ldap_delete_ext**, **ldap_delete_ext_s** **ldap_modify**, **ldap_modify_s** **ldap_modify_ext**, **ldap_modify_ext_s** **ldap_modrdn**, **ldap_modrdn_s** **ldap_rename**, **ldap_rename_s** **ldap_result** **ldap_sasl_bind**, **ldap_sasl_bind_s** **ldap_search**, **ldap_search_s**, **ldap_search_st** **ldap_search_ext**, **ldap_search_ext_s** **ldap_simple_bind**, **ldap_simple_bind_s**
Error Handling	**ldap_err2string** **ldap_get_errno** **ldap_perror** **ldap_result2error**

Table 1. LDAP API Functions (continued)

Category	Function name
Results Processing	**ldap_count_attributes**, **ldap_first_attribute**, **ldap_next_attribute** **ldap_count_entries**, **ldap_first_entry**, **ldap_next_entry** **ldap_count_messages**, **ldap_count_references** **ldap_count_values**, **ldap_get_values** **ldap_count_values_len**, **ldap_get_values_len** **ldap_first_message**, **ldap_first_reference** **ldap_get_dn**, **ldap_get_entry_controls_np** **ldap_explode_dn** **ldap_msgid**, **ldap_msgtype** **ldap_next_message**, **ldap_next_reference** **ldap_parse_result**, **ldap_parse_reference_np** **ldap_parse_sasl_bind_result**
LDAP URL Processing	**ldap_is_ldap_url** **ldap_url_parse** **ldap_url_search**, **ldap_url_search_s**, **ldap_url_search_st**
Utility Functions	**ldap_control_free**, **ldap_controls_free** **ldap_memfree**, **ldap_msgfree** **ldap_mods_free**, **ldap_free_urldesc** **ldap_set_option**, **ldap_set_option_np**, **ldap_get_option** **ldap_set_rebind_proc** **ldap_value_free**, **ldap_value_free_len**

Following is a description of each type of function:

Initialization and Termination Functions
The initialization functions are used to initialize the LDAP programming interface.

Primitive Operations
Each primitive operation comes in two forms, an asynchronous as well as a synchronous form. The synchronous form of the operation is specified by the functions that have the **_s** suffix. An asynchronous LDAP operation allows multiple operations to be initiated by the client program without waiting for the completion of each individual operation. The results of these asynchronous operations are obtained by calling **ldap_result**. The synchronous form of the operation initiates the operation, waits for results, and returns the results to the caller once the results are returned from the server.

Note that **ldap_search** provides the capability to read a single entry, list the sub-entries below a given entry, and search whole sub-trees below a given entry. In this way, all the primitive operations allowed by the XDS programming interface are supported by the LDAP API.

Error Handling Functions
The error handling functions allow for extracting (and displaying) textual information about any LDAP error code that may be returned to the application program.

Results Processing Functions
The results processing functions are all used to interpret the results that come back from an **ldap_search** operation.

LDAP URL Processing Functions
The LDAP URL processing functions work with LDAP-style URLs as specified in RFC 1959 *An LDAP URL Format*. An LDAP URL can specify the parameters necessary to perform an LDAP search operation. These routines parse or use an LDAP URL to perform an LDAP search operation.

Utility Functions
Utility functions are provided for freeing storage that was allocated by the LDAP API on behalf of the caller as well as for setting options that determine certain runtime characteristics of the LDAP

programming interface. An example of an option that can be set is the debug level which allows tracing to be selectively enabled and disabled at run time.

Using the Socksified Client

The z/OS LDAP C/C++ language client can be used to contact LDAP servers through a SOCKS server. The LDAP client has been ″socksified″ so that SOCKS Version 4 (V4) servers can be used to connect to LDAP servers across firewalls on which a SOCKS V4 server is running. The code was developed by the IBM Corporation, the University of California, Berkeley, and NEC Systems Laboratory.

In order to connect to an LDAP server through a SOCKS V4 server, the LDAP client must be provided the location of the SOCKS server or servers in your environment. This can be done in one of two ways:
- Through environment variable settings
- Through environment variable settings along with a SOCKS configuration file (**socks.conf**).

Using only environment variables, the **SOCKS_SERVER** and **RESOLVER_CONFIG** environment variables must be specified in the environment prior to invoking the **ldap_init**, **ldap_ssl_init**, or **ldap_open** LDAP APIs. Using environment variables along with a SOCKS configuration file, the **SOCKS_CONF** and **RESOLVER_CONFIG** environment variables must be specified in the environment prior to invoking the **ldap_init**, **ldap_ssl_init**, or **ldap_open** LDAP APIs.

The **RESOLVER_CONFIG** environment variable specifies the shared domain name server dataset. Refer to *z/OS: Communications Server: IP Configuration Guide*, SC31-8775 for details on specifying the **RESOLVER_CONFIG** environment variable. This environment variable is required in order for Domain Name Service (DNS) to Internet Protocol (IP) address look-ups (**gethostbyname** calls) to work in the environment.

Using the **SOCKS_SERVER** environment variable allows an application that uses the LDAP APIs to specify the location of the SOCKS V4 server to use in connecting to LDAP servers through the SOCKS server. The format for the **SOCKS_SERVER** environment variable value is:

```
export SOCKS_SERVER=9.14.33.90
```

or

```
export SOCKS_SERVER=mysockserver.mycompany.com:1075
```

Using the **SOCKS_CONF** environment variable allows you to make use of a SOCKS configuration file to consolidate the specification of the SOCKS server in your environment. Following is an example of the format for the **SOCKS_CONF** environment variable:

```
export SOCKS_CONF=/home/scott/socks.conf
```

There are three keywords that may be used in the SOCKS configuration file:
- The **sockd** keyword tells the SOCKS client which SOCKS server or servers to use.
- The **deny** keyword tells the SOCKS client which IP address or addresses it should refuse.
- The **direct** keyword tells the SOCKS client that it should bypass the SOCKS server for the given IP address or addresses.

When using the configuration file, the first matching line is used. Therefore, if you list your **sockd** keyword before your **direct** or **deny** keywords, all connections will go through the SOCKS server even though there is another matching line in the configuration file.

If the **SOCKS_SERVER** and **SOCKS_CONF** environment variables are not set, all connections are assumed to be direct.

The format of a **socks.conf** file is shown in Figure 1 on page 6.

Figure 1. Sample socks.conf File

```
##############################################################################
 # Sample SOCKS Configuration File
 #
 # Configuration information is read from the SOCKD.CONF file. Entirely blank
 # lines are ignored. Lines which have a # in the first column are also ignored.
 #
 #
 #   DENY        dst_addr dst_mask
 #   DIRECT      dst_addr dst_mask
 #   SOCKD       {@=serverlist} dst_addr dst_mask
 # Where:
 #   dst_addr    is a dotted quad IP address
 #   dst_mask    is a dotted quad IP address
 #   serverlist  is a comma separated list containing the name or IP addresses
 #               of SOCKS V4 servers (use IP address for speed). Each address
 #               or name may be optionally followed by an explicit port number
 #               as follows:
 #                     IPaddress:portNumber or name:portNumber
 #               Note that the default port number is 1080.
 #               For example, to use port 1081:
 #                   192.168.100.205:1081 or
 #                   mysocksserver:1081
 #
 #
 # On connect, each line is processed in order and the first line that matches
 # is used. If no line matches, the address is assumed to be Direct.
 # In order to cause all non-specific addresses to fail, place the
 # following line at the end of the file:
 #   DENY        0.0.0.0 0.0.0.0
 #
 # Matching is done by taking the destination address and ANDing it with the
 # dst_mask. The result is then compared to the dst_addr. If they match, then
 # if the userlist exists, the current username is compared against this list.
 #
 #   Note:      In this example we are on network 192.168.100.x and the
 #              socks server is on the 192.168.100.205 system. All LDAP
 #              traffic to systems on the 192.168.100 net will be connected
 #              directly, while traffic to all other addresses will be
 #              through the SOCKS server.

##############################################################################
     DIRECT     192.168.100.0 255.255.255.0
     SOCKD      @=192.168.100.205 0.0.0.0 0.0.0.0
 ######################## END OF  FILE #################################################
```

Compiling, Linking, and Running a Program

As previously stated, the LDAP programming interface is provided in a set of C/C++ DLLs. The DLLs will be loaded at program run time so that calls to the functions in the interface can be made. In order to compile and link-edit a program that uses the LDAP API, follow these guidelines:

1. Put

   ```
   #include <ldap.h>
   ```

   in all C or C++ source files that make calls to the LDAP programming interface.
2. When compiling, be sure to specify **-D_OPEN_THREADS** on the compile of the modules that include <ldap.h>.

3. When compiling, be sure to specify **-W0,DLL** on the compile of the modules that make calls to the LDAP API.
4. Be sure your application has **POSIX(ON)** so it can use the LDAP client APIs.
5. When link-editing, be sure to specify the LDAP ″exports″ file in the set of files to be link-edited with the program. When compiling a program to run under the z/OS shell or to run from a PDS, this exports file should be specified as **/usr/lib/GLDCLDAP.x**.

   Note: OS/390 Release 7 of the LDAP server was the last release in which **EUVCLDAP.x** and **EUVCLDAP** were available. Applications should use **GLDCLDAP.x** for link-edit and use **GLDCLDAP** at run time.
6. When running the program, be sure that the LDAP DLL is accessible. When running under the z/OS shell, be sure that the LIBPATH environment variable includes **/usr/lib**. When running the program from an z/OS dataset, the DLLs will be found in **LPALIB**.
7. If using SSL, follow these steps
   a. Put

      ```
      #include <ldapssl.h>
      ```

      in the C/C++ source files that include **ldap.h**.
   b. Ensure that **STEPLIB** or **LIBPATH** identifies the SGSKLOAD DLL.

Here is an example of a Makefile that is used to build the LDAP example program which deletes an LDAP entry. It shows one method of setting up the proper environment for building applications that use the LDAP programming interface:

```
CFLAGS = -g -W0,DLL -D_OPEN_THREADS -Dmvs -DSSL
CC = c89

ldapdelete : ldapdelete.o
    c89 -g -o ldapdelete ldapdelete.o /usr/lib/GLDCLDAP.x

LDAPDLET: ldapdelete.o
    c89 -g -o "//'USER.LOAD(LDAPDLET)'" ldapdelete.o /usr/lib/GLDCLDAP.x
    touch LDAPDLET
```

Using TSO and Batch Jobs

If you are using TSO and batch jobs to compile, link, and run LDAP client applications, you need to be aware of the following additional information:

- Library **SGLDHDRC** (PDS) contains the header files LDAP and LBER (corresponds to HFS file names **ldap.h** and **lber.h**) that are needed to compile LDAP client applications.
- Library **SGLDEXPC** (PDS) contains the export or side-deck file **GLDCLDPX** (corresponds to HFS file name **GLDCLDAP.x**) that is needed by the pre-linker to resolve LDAP DLL function calls. At run time, the LDAP functions are obtained from **LPALIB** module **GLDCLDAP**.
- For the C compile step, the following compiler options are needed:

  ```
  CPARM='LO,DLL,RENT,MARGINS(1,80),NOSEQ,DEF(SSL)'
  ```

  Note: The `MARGINS(1,80)` and `NOSEQ` is needed because **SGLDHDRC(LDAP)** contains source lines that extend into columns 73-80. If sequence numbers are present in the C program source then it is necessary to manually update **SGLDHDRC(LDAP)**.
- The following items are also needed, and it is suggested that they be made part of the C source code:

  ```
  #pragma runopts(POSIX(ON))
  #define mvs
  #define _OPEN_THREADS
  #define MVS_PTHREADS
  #define _OE_SOCKETS
  ```

```
#define _SHARE_EXT_VARS
#define LOCALCP_TRANSLATION
#define EBCDIC_PLATFORM
#define LONGMAP
```

- It is necessary to process the compiler output with the pre-linker to resolve the references to the functions that are in the LDAP DLL. For the pre-link step, specify the PARM **OMVS**. Also at pre-link time, INCLUDE member **GLDCLDPX** from **SGLDEXPC**, for example:

```
//PLKED.SYSLIB   DD  DSN=GLD.SGLDEXPC,DISP=SHR
//PLKED.SYSIN2 DD *
 INCLUDE SYSLIB(GLDCLDPX)
/*
```

Using the API

Using the LDAP programming interface is relatively easy compared to using the XDS/XOM programming interface. Where the XDS/XOM interfaces required setting up some complex nested arrays of XOM structures, many of the parameters for LDAP APIs are simplified to null-terminated character strings. The following sections describe each of the basic parts of a program that uses the LDAP programming interface.

Basic Structure

The basic structure of a program that uses the LDAP programming interface is the following:

1. Initialize the LDAP programming interface and the connection to the directory server that accepts the LDAP protocol using **ldap_open()** or **ldap_init()**.

   An example call to **ldap_open()** looks like:

   ```
   LDAP *ld = ldap_open( "yourhost.acmeInternational.com",
                         LDAP_PORT );
   ```

   The first parameter specifies the DNS host name where the directory server is running and the second parameter specifies the TCP/IP port number that the directory server is listening on for LDAP requests. Port 389 is the default port assigned for LDAP communication. The identifier **LDAP_PORT** is set to 389.

2. Bind to the Directory Service to establish an identity with the directory server by using **ldap_bind()**.

   An example call to **ldap_bind_s()** looks like:

   ```
   rc = ldap_bind_s( ld,
                     "cn=Jane Doe, ou=Marketing, o=Acme International, c=US",
                  password,
                  LDAP_AUTH_SIMPLE );
   ```

   where `password` is a null-terminated character string presumably obtained from the user. The LDAP handle returned from the **ldap_open()** call is used as the first parameter to the **ldap_bind_s()** operation.

3. Perform LDAP operations such as add, modify, delete, compare and search using **ldap_add(_s)**, **ldap_modify(_s)**, **ldap_delete(_s)**, **ldap_compare(_s)**, and **ldap_search(_s)** along with calls to **ldap_result()** for obtaining results from asynchronous operations. Also, interpret the results obtained using the LDAP results processing routines. When using LDAP Version 3 protocol, **ldap_add_ext(_s)**, **ldap_delete_ext(_s)**, **ldap_compare_ext(_s)**, and **ldap_search_ext(_s)** can be used.

   Examples of calls to perform LDAP operations are provided in “Performing an Operation” on page 9. See “Getting Results” on page 12 for examples of calls to **ldap_result()** as well as calls to the LDAP results processing routines. When using LDAP Version 3 protocol, **ldap_parse_result** can be used.

4. When all LDAP operations are completed, unbind and de-initialize the LDAP programming interface using **ldap_unbind()** or **ldap_unbind_s()**. Note that **ldap_unbind_s()** is identical in function to **ldap_unbind()**. It is provided as a convenience to those programs that only do synchronous operations so that the unbind does not appear to be an asynchronous operation. All unbind operations are

synchronous. Also note that after the **ldap_unbind()** or **ldap_unbind_s()** function returns, the LDAP handle that was returned by **ldap_open()** or **ldap_init()** is no longer valid and must not be used.

An example of **ldap_unbind_s()** looks like:

```
rc = ldap_unbind_s( ld );
```

This will unbind (if necessary) from the directory server and de-initialize the LDAP programming interface. After the unbind operation completes, the LDAP handle that was passed into the **ldap_unbind_s()** is no longer valid and must not be used. Its value should be discarded.

Note: In order to terminate the connection with an LDAP server, it is necessary to unbind, regardless of whether an explicit bind (or **ldap_open**) was done.

It is acceptable to perform more than one **ldap_open()** or **ldap_init()** within the same program. More than one LDAP handle can be allocated at the same time. This, however, will cause multiple TCP/IP socket connections to be opened from the client program at the same time. This is discouraged when accessing only one directory server. When multiple directory servers are to be accessed, multiple LDAP handles can be active simultaneously.

Performing an Operation

Each LDAP operation is performed by calling the associated LDAP API. Of the operations, **ldap_add()** and **ldap_modify()** are the most complex to setup while **ldap_search()** is the most complex to interpret the results. It is not surprising that these deal with adding (or changing) and retrieving directory entry contents, respectively.

An example call to each LDAP operation will be shown here along with a short explanation of each parameter's meaning. Refer to "Chapter 2. LDAP Routines" on page 17 for details on the parameters to each LDAP function in the LDAP API.

Example: Adding an Entry

```
modifications = (LDAPMod **)malloc( sizeof(LDAPMod *)*4 );
for( i=0; i<3; i++ ) {
   modifications[i] = (LDAPMod *)malloc( sizeof(LDAPMod) );
   modifications[i]->mod_op = LDAP_MOD_ADD;
}
modifications[3] = NULL;

modifications[0]->mod_type = "objectClass";
modifications[0]->mod_values = (char **)malloc( sizeof(char *)*2 );
modifications[0]->mod_values[0] = "person";
modifications[0]->mod_values[1] = NULL;

modifications[1]->mod_type = "cn";
modifications[1]->mod_values = (char **)malloc( sizeof(char *)*2 );
modifications[1]->mod_values[0] = "John Doe";
modifications[1]->mod_values[1] = NULL;

modifications[2]->mod_type = "sn";
modifications[2]->mod_values = (char **)malloc( sizeof(char *)*2 );
modifications[2]->mod_values[0] = "Doe";
modifications[2]->mod_values[1] = NULL;
rc = ldap_add_s( ld,
                 "cn=John Doe, ou=Marketing, o=Acme International, c=US",
                 modifications );
```

The bulk of the work in calling **ldap_add_s()** is in setting up the modifications array. Once this array is constructed, the call to **ldap_add_s()** is relatively simple. The modifications array represents all the attributes (and associated values) that are to be present in the newly created entry. Note that if a binary attribute value needs to be supplied, the pointer/length form of input should be used. In this case the

mod_op field of the attribute should be set to (**LDAP_MOD_ADD** ; **LDAP_MOD_BVALUES**). This indicates that the value passed in is binary and in pointer/length form.

When data is supplied in a null-terminated character string, it is assumed to be data in the codeset of the current locale. This data will be converted to ASCII (ISO8859-1) prior to being passed to the LDAP server. No conversions are performed on values supplied in pointer/length format. The exception to this is when the **LDAP_OPT_UTF8_IO** option is set to **LDAP_OPT_ON**. In this case, all null-terminated strings are assumed to be UTF-8 strings on input and no translation is performed.

Example: Modifying an Entry

```
modifications = (LDAPMod **)malloc( sizeof(LDAPMod *)*4 );
for( i=0; i<3; i++ ) {
   modifications[i] = (LDAPMod *)malloc( sizeof(LDAPMod) );
}
modifications[3] = NULL;

modifications[0]->mod_op = LDAP_MOD_DELETE;
modifications[0]->mod_type = "sn";
modifications[0]->mod_values = (char **)malloc( sizeof(char *) );
modifications[0]->mod_values[0] = NULL;

modifications[1]->mod_op = LDAP_MOD_ADD;
modifications[1]->mod_type = "email";
modifications[1]->mod_values = (char **)malloc( sizeof(char *)*2 );
modifications[1]->mod_values[0] = "johnd@acme.com";
modifications[1]->mod_values[1] = NULL;

modifications[2]->mod_op = LDAP_MOD_REPLACE;
modifications[2]->mod_type = "email";
modifications[2]->mod_values = (char **)malloc( sizeof(char *)*2 );
modifications[2]->mod_values[0] = "johnd@acmeInternational.com";
modifications[2]->mod_values[1] = NULL;
rc = ldap_modify_s( ld,
                    "cn=John Doe, ou=Marketing, o=Acme International, c=US",
                    modifications );
```

The same modifications array construct that was used for an add operation is used for performing a modify operation. The difference is that the mod_op field can take on values of **LDAP_MOD_ADD**, **LDAP_MOD_CHANGE**, or **LDAP_MOD_DELETE**. Just as for **ldap_add()**, **LDAP_MOD_BVALUES** can be bitwise ORed onto the mod_op field to indicate that binary values are supplied. The same conversion rules are applicable for **ldap_modify()** as were described for **ldap_add()**.

Example: Deleting an Entire Entry

```
msgid = ldap_delete( ld,
                     "cn=John Doe, ou=Marketing, o=Acme International, c=US" );
msgtype = ldap_result( ld, msgid, 1, NULL, &res );
```

It is important to note that the delete operation will fail if the entry to be deleted contains any sub-entries below it in the directory hierarchy. Deletion is not recursive. The example shows how the message ID that is returned from the asynchronous call is passed to the **ldap_result()** function in order to wait for the results of the operation.

Example: Changing the RDN of an Entry

```
rc = ldap_modrdn_s( ld,
                    "cn=John Doe, ou=Marketing, o=Acme International, c=US",
                    "cn=Jonathan Doe",
                    1 );
```

Here, the RDN of the entry is changed. The X.500 data model states that the attribute types and values that comprise the RDN of an entry are also part of the attribute types and values of the entry itself. When the RDN of an entry is modified, it is the option of the program to specify whether the attribute values that made up the old RDN be retained as attribute types and values of the renamed entry. The fourth parameter is used to make this specification. In the example, the old RDN value is deleted.

Example: Comparing an Attribute Value with Its Value in an Entry in the Directory

```
rc = ldap_compare_s( ld,
                     "cn=Jonathan Doe, ou=Marketing, o=Acme International, c=US",
                     "email",
                     "johnd@acmeInternational.com" );
```

This operation compared the supplied value (`"johnd@acmeInternational.com"`) to all the values of the `"email"` attribute in the entry

```
"cn=Jonathan Doe, ou=Marketing, o=Acme International, c=US"
```

If any of the values match, **LDAP_COMPARE_TRUE** is returned. If none of the `"email"` attribute's values match, then **LDAP_COMPARE_FALSE** is returned. If the attribute does not exist or some other error occurs, an appropriate error code is returned.

Example: Reading a Directory Entry's Contents

```
rc = ldap_search_s( ld,
                    "ou=Marketing, o=Acme International, c=US",
                    LDAP_SCOPE_BASE,
                    "(objectClass=*)",
                    NULL, 0, &res );
```

Example: Listing all Sub-Entries of an Entry with Associated objectClass Attribute Values

```
attrs[0] = "objectClass";
attrs[1] = NULL;
rc = ldap_search_s( ld,
                    "ou=Marketing, o=Acme International, c=US",
                    LDAP_SCOPE_ONELEVEL,
                    "(objectClass=*)",
                    attrs, 0, &res );
```

Example: Reading all ObjectClass Attribute Values for all Entries Below a Given Entry

```
attrs[0] = "objectClass";
attrs[1] = NULL;
rc = ldap_search_s( ld,
                    "ou=Marketing, o=Acme International, c=US",
                    LDAP_SCOPE_SUBTREE,
                    "(objectClass=*)",
                    attrs, 0, &res );
```

The **ldap_search_s()** operations shown above exemplify a read, list, and search operation respectively, all by using the **ldap_search_s()** programming interface. In the case of the list operation, the **ldap_get_dn()** function can be used when looping over the returned results to extract just the distinguished name of the sub-entries. Specifying NULL for the attributes parameter will result in all attribute types and values being returned in the results sent to the client program.

Getting Results

The LDAP results processing functions can be used to interpret the results returned from LDAP search operations. Recall that the LDAP search operation is used to perform read and list operations as well. When interpreting the results of a search operation it is usually necessary to loop over the returned entries, for each entry loop over the set of returned attributes, and for each attribute, get the set of attribute values for the attribute. The code to perform this results interpretation takes on a similar format in each case.

An example of this type of processing is:

```
rc = ldap_search_s( ld,
                    "ou=Marketing, o=Acme International, c=US",
                    LDAP_SCOPE_SUBTREE,
                    "(;(cn=Jane*)(cn=Jon*))",
                    NULL, 0, &res );
for( entry = ldap_first_entry( ld, res );
     entry!=NULL;
     entry = ldap_next_entry( ld, entry ) ) {
   dn = ldap_get_dn( ld, entry );
   printf( "Entry: %s\n", dn );
   ldap_memfree( dn );
   for( attrtype = ldap_first_attribute( ld, entry, &ber );
        attrtype != NULL;
        attrtype = ldap_next_attribute( ld, entry, ber ) ) {
      values = ldap_get_values( ld, entry, attrtype );
      i=0;
      while( values[i] != NULL ) {
         printf( "   %s = %s\n", attrtype, values[i] );
         i++;
      }
      ldap_value_free( values );
      ldap_memfree( attrtype );
   }
}
```

As shown by the code fragment, after getting to the attribute type and values for the returned entry, null-terminated character strings are used to represent the attribute type and values. This greatly simplifies accessing Directory Service information.

The **ldap_get_values()** operation provides attribute values in the form of a null-terminated string. This routine will convert the returned results into a null-terminated string in the codeset of the current locale. The data is assured to be (ISO8859-1) coming from the LDAP server. If the data is binary data or conversions should be avoided then the **ldap_get_values_len()** must be used. Data is supplied in pointer/length format and no conversions are performed.

Error Processing

There are four functions in the LDAP programming interface for handling errors returned from LDAP operations. Each is used for a slightly different purpose but all accomplish the same goal of returning error information to the calling program.

Using ldap_get_errno() and ldap_result2error()

The most basic error handling function in the LDAP API is **ldap_get_errno()**. This function simply returns the most recent error condition that was logged by the LDAP programming interface against a given LDAP handle. In the case of LDAP operations that result in errors, the error code value that was returned from the directory server can be obtained by calling **ldap_result2error()**, passing in the LDAPMessage that was returned from the LDAP operation.

There is a subtle difference between using **ldap_get_errno()** and **ldap_result2error()** for asynchronous operations. For asynchronous operations, if an error occurs during the process of sending the request to the directory server, you must use **ldap_get_errno()** to obtain the error value. Use the **ldap_result2error()** call after a **ldap_result(**) call has completed. In the case of synchronous operations, either function can be used. In addition, the synchronous functions also return the error code value for the programmer's convenience.

Be careful in a multi-threaded environment when interpreting the error code. If an LDAP operation completes on a separate thread before the error code value is examined on the current thread, the error code value returned by **ldap_get_errno(**) will be set to the result of the LDAP operation on the other thread. Use the **ldap_result2error()** call in these cases.

Example: Retrieving the Error Code of an Asynchronous Operation Request

```
msgid = ldap_delete( ld,
                     "cn=John Doe, ou=Marketing, o=Acme International, c=US" );
rc = ldap_get_errno( ld );
if ( rc != LDAP_SUCCESS ) {

   /* process the error */

}
```

Example: Retrieving the Error Code Using ldap_result2error()

```
msgtype = ldap_result( ld, msgid, 1, NULL, &res );
rc = ldap_result2error( ld, res, 0 );
if ( rc != LDAP_SUCCESS ) {

   /* process the error */

}
```

Using ldap_err2string() and ldap_perror()

The **ldap_err2string()** function will, given an LDAP error code, return a null-terminated character string that provides a textual description of the error.

Another function available in the LDAP programming interface is **ldap_perror()**. This function will obtain the LDAP error code and issue a message containing the text returned by **ldap_err2string()** on the standard error stream. Note that **ldap_perror()** will send output to the standard error stream even if the LDAP error code is set to **LDAP_SUCCESS** (successful completion).

Be careful in a multi-threaded environment when using **ldap_perror()**. If an LDAP operation completes on a separate thread before **ldap_perror()** examines the error code value on the current thread, the error text emitted by **ldap_perror()** will reflect the result of the LDAP operation on the other thread. Use the **ldap_result2error()** and **ldap_err2string()** calls in these cases.

Example: Obtaining the Using Character String Representing the Error Code

```
rc = ldap_delete_s( ld,
                    "cn=John Doe, ou=Marketing, o=Acme International, c=US"
);
if ( rc != LDAP_SUCCESS ) {
   char *errString = ldap_err2string( rc );

   /* use the error code in a message or log file entry */

}
```

Example: Sending the Result of an Operation to the Standard Error Stream

```
rc = ldap_delete_s( ld,
                    "cn=John Doe, ou=Marketing, o=Acme International, c=US" );
if ( rc != LDAP_SUCCESS ) {
   ldap_perror( ld, "Error on ldap_delete_s()" );
}
```

Tracing

Tracing can be enabled in the LDAP programming interface. This is done by one of two methods. The first method is to use the **ldap_set_option()** API, specifying the option to be set as **LDAP_OPT_DEBUG**. The second method for enabling tracing is to set the **LDAP_DEBUG** environment variable. The value for **LDAP_DEBUG** should be an integer based on the set of trace classes that the user wishes to enable. Consult the ″Debug Levels″ section of the **ldap.h** header file (“ldap.h” on page 108) for a specification of these trace classes. Note that the **LDAP_DEBUG** environment variable can be used without recompiling the client program and provides a means of enabling tracing without changing the client program. The **ldap_set_option()** call can be used for limiting the areas of client program operation that should be traced. Trace output is put on the standard error stream.

An example of enabling all trace classes using the **LDAP_DEBUG** environment variable (assuming the program is running from the z/OS shell) is to enter:

```
export LDAP_DEBUG=65535
```

on the z/OS shell command line prior to running the client program. An example of enabling all trace classes using the **ldap_set_option()** LDAP API is:

```
rc = ldap_set_option( ld, LDAP_OPT_DEBUG, LDAP_DEBUG_ANY );
```

Note: The example above assumes that **LDAP_OPT_PROTOCOL_VERSION** is set to **LDAP_VERSION2**.

The call to **ldap_set_option()** can occur at any point after calling **ldap_open** or **ldap_init** and prior to calling **ldap_unbind()** or **ldap_unbind_s()**.

Thread Safety

The LDAP programming interface is thread safe. This is currently implemented by serializing all operations that are made against a particular LDAP handle. Multiple operations can be safely initiated from multiple threads in the client program. To have these operations sent to the directory server for possible parallel processing by the server, asynchronous operations must be used. An alternative is to initialize multiple LDAP handles. This alternative is not recommended as it will cause multiple open TCP/IP socket connections between the client program and the directory server.

Synchronous Versus Asynchronous Operation

The asynchronous operations in the LDAP programming interface allow multiple operations to be started from the LDAP client without first waiting for each operation to complete. This can be quite beneficial in allowing multiple outstanding search operations from the client program. Searches which take less time to complete can be returned without waiting for a more complicated search to complete.

However, there is some interplay with the thread safety support. In order to allow LDAP operations to be performed from multiple client program threads, operations are serialized. As **ldap_result()** is an LDAP operation, if an **ldap_result()** is initiated on one client thread, any other **ldap_result()** initiated on another client thread will be held up until the **ldap_result()** on the first thread has completed. So, in order to effectively use asynchronous operations to the advantage of the client program, calls to **ldap_result()** should be formulated to complete as quickly as possible so as not to hold up other LDAP operations

possibly initiated on other threads from being started. It is recommended that calls to **ldap_result()** be made to wait for the first available result instead of waiting for specific results when running in a multi-threaded environment.

With synchronous operations, even though multiple operations can be initiated on separate threads, the thread safety support will serialize these requests at the client, prohibiting these requests from being initiated to the server. To ensure that the operations are initiated to the server, asynchronous operations should be used when running in an environment where multiple client program threads may be making calls to the LDAP programming interface.

Calling the LDAP APIs from Other Languages

In order for a COBOL application to call the C LDAP client APIs, the COBOL application must call a C application which, in turn, invokes the LDAP APIs. However, if the COBOL application is link-edited into a separate load module from a C program that calls the LDAP APIs, then the COBOL load module needs to be either link-edited with a **CEEUOPT** that has **POSIX(ON)**, or **POSIX(ON)** has to be passed to it as a runtime option, which is equivalent. See *z/OS: Language Environment Customization*, SA22-7564 for more information.

LDAP Client for Java

LDAP provides an industry-standard Java programming language interface to the LDAP server directory services through the Java Naming and Directory Interface (JNDI). You can find the information about how to use the LDAP service provider interface (LDAP SPI) for JNDI in the online information in **/usr/lpp/ldap/doc/ldappref.html** that is packaged with the code. Open this file in your browser and choose the ″LDAP Client for Java″ heading in the table of contents.

Chapter 2. LDAP Routines

This chapter describes the Lightweight Directory Access Protocol (LDAP) routines which are grouped according to function. The LDAP routines provide access through TCP/IP to directory services which accept the LDAP protocol.

The following references may be helpful when using the LDAP APIs:

- "Chapter 1. LDAP Programming" on page 1 explains how to write applications using the LDAP APIs.
- "Appendix A. LDAP Header Files" on page 107 describes and shows the contents of the header files.
- "Appendix C. Example Programs" on page 121 shows sample programs that use the LDAP APIs.
- *z/OS: SecureWay Security Server LDAP Server Administration and Use* contains information about the LDAP server.

Following is a summary of the LDAP routines:

ldap_abandon
Abandons an asynchronous LDAP operation that is in progress. (See "ldap_abandon" on page 24.)

ldap_abandon_ext
Abandons an asynchronous operation with controls. (See "ldap_abandon" on page 24.)

ldap_add
Performs an asynchronous LDAP add operation. (See "ldap_add" on page 26.)

ldap_add_ext
Performs an asynchronous LDAP add operation with controls. (See "ldap_add" on page 26.)

ldap_add_ext_s
Performs a synchronous LDAP add operation with controls. (See "ldap_add" on page 26.)

ldap_add_s
Performs a synchronous LDAP add operation. (See "ldap_add" on page 26.)

ldap_bind
Binds to an LDAP server asynchronously in order to perform directory operations. (See "ldap_bind" on page 28.)

ldap_bind_s
Binds to an LDAP server synchronously in order to perform directory operations. (See "ldap_bind" on page 28.)

ldap_compare
Performs an asynchronous LDAP compare operation. (See "ldap_compare" on page 32.)

ldap_compare_ext
Performs an asynchronous LDAP compare operation with controls. (See "ldap_compare" on page 32.)

ldap_compare_ext_s
Performs a synchronous LDAP compare operation with controls. (See "ldap_compare" on page 32.)

ldap_compare_s
Performs a synchronous LDAP compare operation. (See "ldap_compare" on page 32.)

ldap_control_free
Frees a single LDAPControl structure. (See "ldap_memfree" on page 60.)

ldap_controls_free
Frees an array of LDAPControl structures. (See "ldap_memfree" on page 60.)

ldap_count_attributes
Counts the number of attributes in an entry returned as part of a search result. (See “ldap_first_entry/reference” on page 42.)

ldap_count_entries
Retrieves a count of the entries in a chain of search results. (See “ldap_get_dn” on page 45.)

ldap_count_messages
Returns the number of messages in a result chain, as returned by **ldap_result**. (See “ldap_message” on page 61.)

ldap_count_references
Returns the number of continuation references in a chain of search results. (See “ldap_get_dn” on page 45.)

ldap_count_values
Counts the number of values in an array of attribute values. (See “ldap_get_values” on page 46.)

ldap_count_values_len
Counts the number of pointers to values in an array of attribute values. (See “ldap_get_values” on page 46.)

ldap_delete
Performs an asynchronous LDAP delete operation. (See “ldap_delete” on page 34.)

ldap_delete_ext
Performs an asynchronous LDAP delete operation with controls. (See “ldap_delete” on page 34.)

ldap_delete_ext_s
Performs a synchronous LDAP delete operation with controls. (See “ldap_delete” on page 34.)

ldap_delete_s
Performs a synchronous LDAP delete operation. (See “ldap_delete” on page 34.)

ldap_err2string
Provides a textual description of an error message. (See “ldap_error” on page 36.)

ldap_explode_dn
Parses LDAP distinguished names. (See “ldap_get_dn” on page 45.)

ldap_first_attribute
Begins stepping through an LDAP entry’s attributes. (See “ldap_first_entry/reference” on page 42.)

ldap_first_entry
Retrieves the first entry in a chain of search results. (See “ldap_get_dn” on page 45.)

ldap_first_message
Retrieves the first message in a result chain, as returned by **ldap_result**. (See “ldap_message” on page 61.)

ldap_first_reference
Retrieves the first continuation reference in a chain of search results. (See “ldap_get_dn” on page 45.)

ldap_free_urldesc
Deallocates an LDAP URL description obtained from a call to **ldap_url_parse**. (See “ldap_url” on page 81.)

ldap_get_dn
Obtains LDAP distinguished names from an LDAP entry. (See “ldap_get_dn” on page 45.)

ldap_get_entry_controls_np
Extracts server controls from an entry. (See “ldap_get_dn” on page 45.)

ldap_get_errno
Retrieves the last error code set by an LDAP operation. (See “ldap_error” on page 36.)

ldap_get_option
Retrieves the current value of an LDAP option. (See “ldap_init” on page 49.)

ldap_get_values
Retrieves attribute values from an LDAP entry in NULL-terminated character strings. (See “ldap_get_values” on page 46.)

ldap_get_values_len
Retrieves attribute values from an LDAP entry in pointer/length format. (See “ldap_get_values” on page 46.)

ldap_init
Initializes an LDAP context. (See “ldap_init” on page 49.)

ldap_is_ldap_url
Checks whether a character string represents an LDAP Uniform Resource Locator (URL). (See “ldap_url” on page 81.)

ldap_memfree
Deallocates character strings allocated by the LDAP programming interface. (See “ldap_memfree” on page 60.)

ldap_modify
Performs an asynchronous LDAP modify operation. (See “ldap_modify” on page 63.)

ldap_modify_ext
Performs an asynchronous LDAP modify operation with controls. (See “ldap_modify” on page 63.)

ldap_modify_ext_s
Performs a synchronous LDAP modify operation with controls. (See “ldap_modify” on page 63.)

ldap_modify_s
Modifies LDAP entries synchronously. (See “ldap_modify” on page 63.)

ldap_modrdn
Performs an asynchronous LDAP modify relative distinguished name (RDN) operation. (See “ldap_rename” on page 68.)

ldap_modrdn_s
Performs a synchronous LDAP modify RDN operation. (See “ldap_rename” on page 68.)

ldap_mods_free
Deallocates a NULL-terminated array of modification structures. (See “ldap_modify” on page 63.)

ldap_msgfree
Deallocates the memory allocated for a result. (See “ldap_rename” on page 68.)

ldap_msgid
Retrieves the message ID associated with an LDAP message. (See “ldap_result” on page 71.)

ldap_msgtype
Retrieves the next attribute type name in an LDAP result. (See “ldap_result” on page 71.)

ldap_next_attribute
Deallocates a NULL-terminated array of modification structures. (See “ldap_first_entry/reference” on page 42.)

ldap_next_entry
Retrieves the next entry in a chain of search results to parse. (See “ldap_get_dn” on page 45.)

ldap_next_message
Retrieves the next message in a result chain, as returned by **ldap_result**. (See “ldap_message” on page 61.)

ldap_next_reference
Retrieves the next continuation reference in a chain of search results. (See “ldap_get_dn” on page 45.)

ldap_open
Initializes an LDAP context and opens a connection to an LDAP server under that context. (See “ldap_init” on page 49.)

ldap_parse_reference_np
Extracts information from a continuation reference. (See “ldap_get_dn” on page 45.)

ldap_parse_result
Extracts information from results. (See “ldap_parse_result” on page 66.)

ldap_parse_sasl_bind_result
Extracts server credentials from SASL bind results. (See “ldap_parse_result” on page 66.)

ldap_perror
Prints an indication of the error on the standard error stream. (See “ldap_error” on page 36.)

ldap_rename
Performs an asynchronous LDAP rename operation. (See “ldap_rename” on page 68.)

ldap_rename_s
Performs a synchronous LDAP rename operation. (See “ldap_rename” on page 68.)

ldap_result
Waits for the result of an LDAP operation. (See “ldap_result” on page 71.)

ldap_result2error
Interprets a result as returned by **ldap_result** or one of the synchronous LDAP search operation routines. (See “ldap_error” on page 36.)

ldap_sasl_bind
Binds to an LDAP server asynchronously in order to perform directory operations using the Simple Authentication Security Layer (SASL). (See “ldap_bind” on page 28.)

ldap_sasl_bind_s
Binds to an LDAP server synchronously in order to perform directory operations using the Simple Authentication Security Layer (SASL). (See “ldap_bind” on page 28.)

ldap_search
Performs an asynchronous LDAP search operation. (See “ldap_search” on page 73.)

ldap_search_ext
Performs an asynchronous LDAP search operation with controls. (See “ldap_search” on page 73.)

ldap_search_ext_s
Performs a synchronous LDAP search operation with controls. (See “ldap_search” on page 73.)

ldap_search_s
Performs a synchronous LDAP search operation. (See “ldap_search” on page 73.)

ldap_search_st
Performs a synchronous LDAP search operation allowing a time-out to be specified to limit the time to wait for results. (See “ldap_search” on page 73.)

ldap_set_option
Sets the value of an LDAP option. (See “ldap_init” on page 49.)

ldap_set_option_np
Sets the value of an LDAP option. This API is nonportable. (See “ldap_init” on page 49.)

ldap_set_rebind_proc
Establishes a call-back function for rebinding during referrals chasing. (See “ldap_bind” on page 28.)

ldap_simple_bind
Binds to an LDAP server asynchronously using simple authentication in order to perform directory operations. (See “ldap_bind” on page 28.)

ldap_simple_bind_s
Binds to an LDAP server synchronously using simple authentication in order to perform directory operations. (See “ldap_bind” on page 28.)

ldap_ssl_client_init
Initializes the SSL library. (See “ldap_ssl” on page 77.)

ldap_ssl_init
Initializes an SSL connection. (See “ldap_ssl” on page 77.)

ldap_ssl_start
Creates a secure SSL connection. (See “ldap_ssl” on page 77.)

ldap_unbind
Unbinds from an LDAP server asynchronously and deallocates an LDAP handle. (See “ldap_bind” on page 28.)

ldap_unbind_s
Unbinds from an LDAP server synchronously and deallocates an LDAP handle. (See “ldap_bind” on page 28.)

ldap_url_parse
Breaks down an LDAP URL into its component pieces. (See “ldap_url” on page 81.)

ldap_url_search
Initiates an asynchronous LDAP search based on an LDAP URL. (See “ldap_url” on page 81.)

ldap_url_search_s
Initiates a synchronous LDAP search based on an LDAP URL. (See “ldap_url” on page 81.)

ldap_url_search_st
Initiates a synchronous LDAP search based on an LDAP URL allowing a time-out to be specified to limit the time to wait for results. (See “ldap_url” on page 81.)

ldap_value_free
Deallocates values returned by **ldap_get_values**. (See “ldap_get_values” on page 46.)

ldap_value_free_len
Deallocates values returned by **ldap_get_values_len**. (See “ldap_get_values” on page 46.)

LDAP Controls

Certain LDAP Version 3 operations can be extended with the use of *controls*. Controls can be sent to a server, or returned to the client with any LDAP message. This type of control is called a *server control*.

The LDAP API also supports a client-side extension mechanism, which can be used to define *client controls*. The client-side controls affect the behavior of the LDAP client library, and are never sent to the server. Note that client-side controls are not defined for this client library. A common data structure is used to represent both server-side and client-side controls:

```
typedef struct ldapcontrol {
        char *ldctl_oid;
        struct berval ldctl_value;
        char ldctl_iscritical;
} LDAPControl;
```

The LDAPControl fields have the following definitions:

ldct6l_old
: Specifies the control type, presented as a string.

ldctl_value
: Specifies the data associated with the control (if any). To specify a zero-length value, set **ldctl_value.bv_len** to zero and **ldctl_value.bv_val** to a zero-length string. To indicate that no data is associated with the control, set **ldctl_value.bv_val** to NULL.

ldctl_iscritical
: Specifies whether the control is critical. If this field is nonzero (critical), the operation is performed only if the control is appropriate for the operation and it is recognized and supported by the server (or the client for client-side controls). In this case, the control is used in performing the operation.

  If this field is zero (noncritical), the control is used in performing the operation only if it is appropriate for the operation and it is recognized and supported by the server (or the client for client-side controls). Otherwise, the control will be ignored.

Controls are specified on the LDAP API as lists of controls. Control lists are represented as a NULL-terminated array of pointers to LDAPControl structures.

Session Controls

Many of the LDAP Version 3 APIs which perform LDAP operations accept a list of controls (for example, **ldap_search_ext**). Alternatively, a list of controls that affects each operation performed on a given LDAP handle can be set using the **ldap_set_option** API. These are called session controls. Session controls apply to the given operation when NULL is specified for the corresponding control list parameter on the API. If a list of controls is specified for the control parameter on the API, these are used instead of the session controls on the given operation. If session controls are set, but a specific request does not want any controls, an empty list of controls should be specified for the control parameter. (This is different from a NULL parameter; it is a pointer to an array containing a single NULL.)

Session controls also apply to the nonextended APIs which perform LDAP operations. So although **ldap_search**, for example, does not accept control list parameters, it will include a server control on its request if there was a server control set up through **ldap_set_option**.

Using RACF® Data

There are some restrictions when updating information stored in RACF, a component of the SecureWay Security Server for z/OS, over the LDAP protocol. See the information about accessing RACF information in *z/OS: SecureWay Security Server LDAP Server Administration and Use*.

Deprecated LDAP APIs

Although the following APIs are still supported, their use is deprecated. Use of the newer replacement APIs is strongly encouraged:

- **ldap_ssl_start** (use **ldap_ssl_client_init** and **ldap_ssl_init**)
- **ldap_open** (use **ldap_init**)
- **ldap_bind** (use **ldap_simple_bind**)
- **ldap_bind_s** (use **ldap_simple_bind_s**)
- **ldap_modrdn** (use **ldap_rename**)

- **ldap_modrdn_s** (use **ldap_rename_s**)
- **ldap_result2error** (use **ldap_parse_result**)
- **ldap_perror** (use **ldap_parse_result**)

ldap_abandon

ldap_abandon
ldap_abandon_ext

Purpose

Abandon an asynchronous LDAP operation that is in progress.

Format

```
#include <ldap.h>

int ldap_abandon(
        LDAP *ld,
        int msgid)


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

Parameters

Input

ld Specifies the LDAP handle returned by a previous call to **ldap_open**, **ldap_ssl_init**, or **ldap_init**.

msgid
The message ID of an outstanding LDAP operation as returned by a call to an asynchronous operation such as **ldap_search**, **ldap_modify**, and so on.

serverctrls
Specifies a list of LDAP server controls. This parameter may be set to NULL. See “LDAP Controls” on page 21 for more information about server controls.

clientctrls
Specifies a list of LDAP client controls. This parameter may be set to NULL. See “LDAP Controls” on page 21 for more information about client controls.

Usage

The **ldap_abandon** and **ldap_abandon_ext** APIs are used to abandon or cancel an LDAP operation in progress.

Both APIs check to see if the result of the operation has already been returned by the server. If it has, it deletes it from the queue of pending received messages. If not, it sends an LDAP abandon operation to the LDAP server.

The result of an abandoned operation will not be returned from a future call to **ldap_result**.

Session controls set by the **ldap_set_option** API apply to both **ldap_abandon** and **ldap_abandon_ext**. The **ldap_abandon_ext** API allows controls to be specified which override the session controls for the given call.

Error Conditions

The **ldap_abandon** API returns 0 if it is successful, -1 otherwise. Use **ldap_get_errno** to retrieve the error value. See “ldap_error” on page 36 for possible values.

The **ldap_abandon_ext** API returns **LDAP_SUCCESS** if successful, otherwise an error code is returned.

Related Topics

ldap_result
ldap_error

ldap_add

ldap_add
ldap_add_s
ldap_add_ext
ldap_add_ext_s

Purpose

Perform an LDAP add operation.

Format

```
#include <ldap.h>

int ldap_add(
        LDAP *ld,
        char *dn,
        LDAPMod *attrs[])

int ldap_add_s(
        LDAP *ld,
        char *dn,
        LDAPMod *attrs[])

int ldap_add_ext(
        LDAP *ld,
        char *dn,
        LDAPMod *attrs [] ,
        LDAPControl **serverctrls,
        LDAPControl **clientctrls,
        int *msgidp)

int ldap_add_ext_s(
        LDAP *ld,
        char *dn,
        LDAPMod *attrs[],
        LDAPControl **serverctrls,
        LDAPControl **clientctrls)
```

Parameters

Input

ld Specifies the LDAP handle returned by a previous call to **ldap_open**, **ldap_ssl_init**, or **ldap_init**.

dn Specifies the distinguished name of the entry to add.

attrs
A NULL-terminated array of the entry's attributes. The LDAPMod structure is used to represent attributes, with the *mod_type* and *mod_values* fields being used as described under **ldap_modify**, and the *mod_op* field being used only if you need to specify the **LDAP_MOD_BVALUES** option. Otherwise, it should be set to 0. The LDAPMod structure is shown in "ldap_modify" on page 63.

serverctrls
Specifies a list of LDAP server controls. This parameter may be set to NULL. See "LDAP Controls" on page 21 for more information about server controls.

clientctrls
Specifies a list of LDAP client controls. This parameter may be set to NULL. See "LDAP Controls" on page 21 for more information about client controls.

Output

msgidp
This result parameter is set to the message ID of the request if the **ldap_add_ext** API succeeds.

Usage

Note that all entries except that specified by the last component in the given DN must already exist.

When data is supplied in a NULL-terminated character string, it is assumed to be data in the codeset of the current locale. This data will be converted to UTF-8 prior to being passed to the LDAP server. No conversions are performed on values supplied in pointer/length format (that is, those values specified in berval structures and when **LDAP_MOD_BVALUES** is specified).

The **ldap_add_ext** API initiates an asynchronous add operation and returns the constant **LDAP_SUCCESS** if the request was successfully sent, or another LDAP error code if not. If successful, **ldap_add_ext** places the message ID of the request in **msgidp*. A subsequent call to **ldap_result** can be used to obtain the result of the operation. The **ldap_parse_result** API is used to extract information from the result, including any error information.

Similarly, the **ldap_add** API initiates an asynchronous add operation and returns the message ID of the request it initiated. The result of this operation can be obtained by calling **ldap_result**, and result information can be extracted by calling **ldap_parse_result**.

The synchronous **ldap_add_ext_s** and **ldap_add_s** APIs both return the resulting error code of the add operation.

All four of the LDAP add APIs support session controls set by the **ldap_set_option** API. The **ldap_add_ext** and **ldap_add_ext_s** APIs both allow LDAP Version 3 server controls and client controls to be specified with the request which overrides the session controls.

Error Conditions

The **ldap_add** API returns -1 in case of an error initiating the request. Use **ldap_get_errno** to retrieve the error value. See “ldap_error” on page 36 for possible values.

The **ldap_add_s**, **ldap_add_ext**, and **ldap_add_ext_s** APIs return **LDAP_SUCCESS** if successful, otherwise an error code is returned. See “ldap_error” on page 36 for possible values.

If the add is directed to an z/OS LDAP server running with an SDBM database, the **ldap_add** APIs can return **LDAP_OTHER** and have completed a partial update to an entry in RACF. The results will match what would occur if the update were done using the RACF **altuser** command. If several RACF attributes are being updated and one of them is in error, RACF reports on the error, but still updates the other attributes. The RACF message text is also returned in the result.

Related Topics

ldap_modify

ldap_bind

ldap_sasl_bind
ldap_sasl_bind_s
ldap_simple_bind
ldap_simple_bind_s
ldap_unbind
ldap_unbind_s
ldap_set_rebind_proc
ldap_bind (deprecated)
ldap_bind_s (deprecated)

Purpose

LDAP routines for binding and unbinding.

Format

```
#include <ldap.h>

int ldap_sasl_bind(        LDAP *ld,
        char *who,
        char *mechanism,
        struct berval *cred,
        LDAPControl **serverctrls,
        LDAPControl **clientctrls,
        int *msgidp)

int ldap_sasl_bind_s(
        LDAP *ld,
        char *who,
        char *mechanism,
        struct berval *cred,
        LDAPControl **serverctrls,
        LDAPControl **clientctrls,
        struct berval.**servercredp)

int ldap_simple_bind(
        LDAP *ld,
        char *who,
        char *passwd)

int ldap_simple_bind_s(
        LDAP *ld,
        char *who,
        char *passwd)

int ldap_unbind(
        LDAP *ld)

int ldap_unbind_s(
        LDAP *ld)

void ldap_set_rebind_proc(
        LDAP *ld,
        LDAPRebindProc rebindproc)

int ldap_bind(
        LDAP *ld,
        char *who,
        char *cred,
        int method)

int ldap_bind_s(
```

```
LDAP *ld,
char *who,
char *cred,
int method)
```

Parameters

Input

ld Specifies the LDAP handle returned by a previous call to **ldap_open**, **ldap_ssl_init**, or **ldap_init**.

who
Specifies the distinguished name of the entry as which to bind.

cred
Specifies the password used in association with the DN of the entry (*who*) as which to bind for simple authentication. Arbitrary credentials can be passed using this parameter. In most cases, this is the DN's password.

When using a SASL bind, the format and content of the credentials depends on the setting of the *mechanism* parameter.

mechanism
Although a variety of mechanisms have been IANA (Internet Assigned Numbers Authority) registered, the only mechanism supported by the library at this time is the **LDAP_MECHANISM_EXTERNAL** mechanism, represented by the string `LDAP_MECHANISM_EXTERNAL`.

The **LDAP_MECHANISM_EXTERNAL** mechanism indicates to the server that information external to SASL should be used to determine whether the client is authorized to authenticate. For this implementation, the system providing the external information must be SSL. For example, if the client sets *dn* and *credential* to NULL (the value of the pointers should be NULL), with *mechanism* set to **LDAP_MECHANISM_EXTERNAL**, the client is requesting that the server use the strongly authenticated identity to access the directory.

method
Selects the authentication method to use. Specify **LDAP_AUTH_SIMPLE** for simple authentication. (Simple authentication is the only supported method.)

passwd
Specifies the password used in association with the DN of the entry as which to bind.

serverctrls
Specifies a list of LDAP server controls. This parameter may be set to NULL. See "LDAP Controls" on page 21 for more information about server controls.

clientctrls
Specifies a list of LDAP client controls. This parameter may be set to NULL. See "LDAP Controls" on page 21 for more information about client controls.

rebindproc
Specifies the pointer to a function that will be invoked to gather the information necessary to bind to another LDAP server.

Output

msgidp
This result parameter is set to the message ID of the request if the **ldap_sasl_bind** call succeeds.

servercredp
This result parameter is set to the credentials returned by the server. If no credentials are returned, it will be set to **NULL**.

Usage

These APIs provide various interfaces to the LDAP bind operation. After the LDAP handle is initialized with **ldap_init** or is initialized and a connection is made to an LDAP Version 2 server using **ldap_open**, an LDAP bind operation must be performed before other operations can be attempted over the connection. Both synchronous and asynchronous version of each variant of the bind API are provided.

When communicating with an LDAP server that supports the LDAP Version 3 protocol, bind is optional. The absence of a bind will be interpreted by the LDAP Version 3 server as a request for unauthenticated access. A bind is required by LDAP servers that only support the LDAP Version 2 protocol.

Simple Authentication

The simplest form of the bind call is the synchronous API **ldap_simple_bind_s**. It takes the DN to bind as, as well as the password associated with that DN (supplied in *passwd*). It returns an LDAP error indication (see “ldap_error” on page 36). The **ldap_simple_bind** call is asynchronous, taking the same parameters but only initiating the bind operation and returning the message ID of the request it sent. The result of the operation can be obtained by a subsequent call to **ldap_result**.

General Authentication

The **ldap_bind** and **ldap_bind_s** routines are deprecated. They can be used when the authentication method to use needs to be selected at run time. They both take an extra *method* parameter selecting the authentication method to use. However, *method* must be set to **LDAP_AUTH_SIMPLE**, to select simple authentication (the only supported method). The **ldap_bind** returns the message ID of the initiated request. The **ldap_bind_s** API returns an LDAP error indication, or **LDAP_SUCCESS** on successful completion.

SASL Authentication

The **ldap_sasl_bind** and **ldap_sasl_bind_s** APIs can be used to do simple and certificate authentication over LDAP through the use of the Simple Authentication Security Layer (SASL). By setting *mechanism* to **LDAP_SASL_SIMPLE** the SASL bind request will be interpreted as a request for simple authentication (that is, equivalent to using **ldap_simple_bind** or **ldap_simple_bind_s**). By setting *mechanism* to **LDAP_MECHANISM_EXTERNAL**, the SASL bind request will be interpreted as a request for certificate authentication.

With this implementation, the primary reason for using the SASL bind facility is to use the client authentication mechanism provided by SSL to strongly authenticate to the directory server, using the client’s X.509 certificate. For example, the client application can use the following logic:

1. **ldap_ssl_client_init** (initialize the SSL library)
2. **ldap_ssl_init** (*host*, *port*, *name*), where *name* references a public/private key pair in the client’s key ring file
3. **ldap_sasl_bind_s** (*ld*, *who*=`NULL`, *mechanism*=`LDAP_MECHANISM_EXTERNAL`, *cred*=`NULL`...)

A server that supports this mechanism can then access the directory using the strongly authenticated client identity (as extracted from the client’s X.509 certificate).

By setting mechanism to a NULL pointer, the SASL bind request will be interpreted as a request for simple authentication (that is, equivalent to using **ldap_simple_bind** or **ldap_simple_bind_s**).

Unbinding

The **ldap_unbind** API is used to unbind from the directory, terminate the current association, and deallocate the resources associated with the LDAP handle. Once it is called, any open connection to the LDAP server is closed and the LDAP handle is not valid. The **ldap_unbind_s** and **ldap_unbind** APIs are both synchronous, either can be called.

Rebinding While Following Referrals

When the LDAP client is returned a referral to a different LDAP server, it may need to rebind to that server. In order to do this, the client must have the proper credentials available to pass to the target LDAP server. Normally, these credentials are passed on the **ldap_bind** function invocation. During referrals processing, however, this must be done when needed by the LDAP client. The rebind procedure is called twice when attempting to rebind to an LDAP server: once to obtain the credentials for the user and once to allow the rebind procedure to release any storage that was allocated by the first call to the rebind procedure.

The *rebindproc* parameter is a pointer to a function that has the following prototype:

```
int ldapRebindProc(
       LDAP *ld,
       char **dnp,
       char **passwdp,
       int *authmethodp,
       int freeit )
```

When the rebind procedure is invoked and the *freeit* input parameter is zero (0), the rebind procedure should set the *dnp*, *passwdp*, and *authmethodp* fields before returning to the caller. The only supported authentication method for rebinding is **LDAP_AUTH_SIMPLE**. **LDAP_SUCCESS** should be returned if the fields were successfully returned to the caller, otherwise one of the error codes defined in **ldap.h** should be returned by the rebind procedure to the caller. If the return code is not set to **LDAP_SUCCESS**, the operation will be stopped and the specified error code will be returned to the original caller.

When the rebind procedure is invoked and the *freeit* input parameter is nonzero, the rebind procedure should release any storage that was acquired by a previous call to the rebind procedure where the *freeit* parameter was zero. When the *freeit* parameter field is nonzero, the *dnp*, *passwdp*, and *authmethodp* parameters should be treated as input parameters.

If a rebind procedure is not established, then the client library will use unauthenticated access when following referrals to additional servers.

Error Conditions

The **ldap_sasl_bind**, **ldap_simple_bind**, **ldap_unbind**, and **ldap_bind** APIs return -1 in case of an error initiating the request. Use **ldap_get_errno** to retrieve the error value. See “ldap_error” on page 36 for possible values.

The **ldap_sasl_bind_s**, **ldap_simple_bind_s**, **ldap_unbind_s**, and **ldap_bind_s** APIs return **LDAP_SUCCESS** if successful, otherwise an error code is returned. See “ldap_error” on page 36 for possible values.

Related Topics

ldap_open
ldap_error

ldap_compare

ldap_compare
ldap_compare_s
ldap_compare_ext
ldap_compare_ext_s

Purpose

Perform an LDAP compare operation.

Format

```
#include <ldap.h>

typedef struct berval {
    unsigned long bv_len;
    char *bv_val;
};

int ldap_compare(
        LDAP *ld,
        char *dn,
        char *attr,
        char *value)


int ldap_compare_s(
        LDAP *ld,
        char *dn,
        char *attr,
        char *value)


int ldap_compare_ext(
        LDAP *ld,
        char *dn,
        char *attr,
        struct berval *bvalue,
        LDAPControl **serverctrls,
        LDAPControl **clientctrls,
        int *msgidp)


int ldap_compare_ext_s(
        LDAP *ld,
        char *dn,
        char *attr,
        struct berval *bvalue,
        LDAPControl **serverctrls,
        LDAPControl **clientctrls)
```

Parameters

Input

ld Specifies the LDAP handle returned by a previous call to **ldap_open**, **ldap_ssl_init**, or **ldap_init**.

dn Specifies the distinguished name of the entry upon which to perform the compare.

attr
Specifies the attribute type to compare to the attribute found in the entry.

bvalue
Specifies the attribute value to compare against the value in the entry. This parameter is used in the

ldap_compare_ext and **ldap_compare_ext_s** APIs, and is a pointer to a berval structure (see “ldap_get_values” on page 46), and is used to compare binary values.

value
Specifies the attribute value to compare to the value found in the entry. This parameter is used in the **ldap_compare** and **ldap_compare_s** APIs, and is used to compare string attributes. Use **ldap_compare_ext** or **ldap_compare_ext_s** if you need to compare binary values.

serverctrls
Specifies a list of LDAP server controls. This parameter may be set to NULL. See “LDAP Controls” on page 21 for more information about server controls.

clientctrls
Specifies a list of LDAP client controls. This parameter may be set to NULL. See “LDAP Controls” on page 21 for more information about client controls.

Output

msgidp
This result parameter is set to the message ID of the request if the **ldap_compare_ext** API succeeds.

Usage

The **ldap_compare_ext** API initiates an asynchronous compare operation and returns the constant **LDAP_SUCCESS** if the request was successfully sent, or another LDAP error code if not. If successful, **ldap_compare_ext** places the message ID of the request in **msgidp*. A subsequent call to **ldap_result** can be used to obtain the result of the operation. The **ldap_parse_result** API is used to extract information from the result, including any error information. The error code indicates if the operation completed successfully (**LDAP_COMPARE_TRUE** or **LDAP_COMPARE_FALSE**). Any other error code indicates a failure performing the operation.

Similarly, the **ldap_compare** API initiates an asynchronous compare operation and returns the message ID of the request it initiated. The result of the compare can be obtained by a subsequent call to **ldap_result**, and result information can be extracted by calling **ldap_parse_result**.

The synchronous **ldap_compare_s** and **ldap_compare_ext_s** APIs both return the resulting error code of the compare operation.

All four of the LDAP compare APIs support session controls set by the **ldap_set_option** API. The **ldap_compare_ext** and **ldap_compare_ext_s** APIs both allow LDAP Version 3 server controls and client controls to be specified with the request which overrides the session controls.

Error Conditions

The **ldap_compare** API returns -1 in case of an error initiating the request. Use **ldap_get_errno** to retrieve the error value. See “ldap_error” on page 36 for possible values.

The **ldap_compare_s** API returns **LDAP_COMPARE_TRUE** (if the entry contains the attribute value) or **LDAP_COMPARE_FALSE** (if the entry does not contain the attribute value) if successful, otherwise an error code is returned. See “ldap_error” on page 36 for possible values.

Related Topics

ldap_error

ldap_delete

ldap_delete
ldap_delete_s
ldap_delete_ext
ldap_delete_ext_s

Purpose

Perform an LDAP delete operation.

Format

```
#include <ldap.h>

int ldap_delete(
        LDAP *ld,
        char *dn)


int ldap_delete_s(
        LDAP *ld,
        char *dn)


int ldap_delete_ext(
        LDAP *ld,
        char *dn,
        LDAPControl **serverctrls,
        LDAPControl **clientctrls,
        int *msgidp)


int ldap_delete_ext_s(
        LDAP *ld,
        char *dn,
        LDAPControl **serverctrls,
        LDAPControl **clientctrls)
```

Parameters

Input

ld Specifies the LDAP handle returned by a previous call to **ldap_open**, **ldap_ssl_init**, or **ldap_init**.

dn Specifies the distinguished name of the entry to be deleted.

serverctrls
Specifies a list of LDAP server controls. This parameter may be set to NULL. See “LDAP Controls” on page 21 for more information about server controls.

clientctrls
Specifies a list of LDAP client controls. This parameter may be set to NULL. See “LDAP Controls” on page 21 for more information about client controls.

Input

msgidp
This result parameter is set to the message ID of the request if the **ldap_delete_ext** API succeeds.

Usage

Note that the entry to delete must be a leaf entry (that is, it must not have any children). Deletion of entire subtrees in a single operation is not supported by LDAP. However, the **sdelete** example program provides

example code on how deletion of a subtree of LDAP entries could be performed. The example programs can be found in the **/usr/lpp/ldap/examples** directory.

The **ldap_delete_ext** API initiates an asynchronous delete operation and returns the constant **LDAP_SUCCESS** if the request was successfully sent, or another LDAP error code if not. If successful, **ldap_delete_ext** places the message ID of the request in **msgidp*. A subsequent call to **ldap_result** can be used to obtain the result of the operation. The **ldap_parse_result** API is used to extract information from the result, including any error information. The error code indicates if the operation completed successfully. The **ldap_parse_result** API is used to check the error code in the result.

Similarly, the **ldap_delete** API initiates an asynchronous delete operation and returns the message ID of the request it initiated. The result of the delete can be obtained by a subsequent call to **ldap_result**, and result information can be extracted by calling **ldap_parse_result**.

The synchronous **ldap_delete_s** and **ldap_delete_ext_s** perform LDAP delete operations and both return the resulting error code of the compare operation.

All four of the LDAP delete APIs support session controls set by the **ldap_set_option** API. The **ldap_delete_ext** and **ldap_delete_ext_s** APIs both allow LDAP Version 3 server controls and client controls to be specified with the request which overrides the session controls.

Error Conditions

The **ldap_delete** API returns -1 in case of an error initiating the request. Use **ldap_get_errno** to retrieve the error value. See “ldap_error” on page 36 for possible values.

The **ldap_delete_s** API returns **LDAP_SUCCESS** if successful, otherwise an error code is returned. See “ldap_error” on page 36 for possible values.

Related Topics

ldap_error

ldap_error

ldap_get_errno
ldap_perror (deprecated)
ldap_result2error (deprecated)
ldap_err2string

Purpose

LDAP protocol error handling routines.

Format

```
#include <ldap.h>

int ldap_get_errno(
        LDAP *ld )


void ldap_perror(
        LDAP *ld,
        char *s)


int ldap_result2error(
        LDAP *ld,
        LDAPMessage *res,
        int freeit)


char *ldap_err2string(
        int err)
```

Parameters

Input

ld Specifies the LDAP handle returned by a previous call to **ldap_open**, **ldap_ssl_init**, or **ldap_init**.

s Specifies the message prefix, which is prepended to the string form of the error code held stored under the LDAP handle. The string form of the error is the same string that would be returned by a call to **ldap_err2string**.

res
Specifies an LDAP result that was returned by a previous call to **ldap_result** or one of the synchronous LDAP search routines (see “ldap_search” on page 73).

freeit
Specifies whether to deallocate the *res* LDAP result. If nonzero, the *res* parameter is deallocated as part of the call to **ldap_result2error**.

err Specifies the error to be described.

Usage

These APIs provide interpretation of the various error codes returned by the LDAP protocol and LDAP library APIs.

It is sometimes inconvenient to pass the return code of an LDAP operation back to the caller in the case of an error. Further, for asynchronous LDAP operations, no error code is returned by the call. In each of these cases, the **ldap_get_errno** API can be used to retrieve the last set error code for the LDAP handle that is passed on input.

Note: In multi-threaded applications, the value returned by the **ldap_get_errno** routine is the last error set by the last LDAP operation performed against the LDAP handle. It is possible for an LDAP operation on a different thread to reset the error value stored under the LDAP handle before the original error code is retrieved.

The **ldap_perror** API prints the message prefix followed by the result of a call to **ldap_err2string** (**ldap_get_errno(***ld***)**) to the standard error stream.

Note: In multi-threaded applications, the error text printed corresponds to the last error value set by the last LDAP operation performed against the LDAP handle. It is possible for an LDAP operation on a different thread to reset the error value stored under the LDAP handle before the original error text is retrieved.

The **ldap_result2error** API takes *res*, a result as produced by **ldap_result**, or the synchronous LDAP search operation routines and returns the corresponding error code.

The **ldap_err2string** API provides interpretation of the various error codes returned by the LDAP protocol and LDAP library routines and returned by the **ldap_get_errno** API.

The **ldap_err2string** API is used to convert the numeric LDAP error code, as returned by **ldap_parse_result** or **ldap_parse_sasl_bind_result**, or one of the synchronous APIs, into a NULL-terminated character string that describes the error. Do not modify or attempt to deallocate this string.

Error Conditions

The possible values for an LDAP error code are listed in the following table.

Table 2. LDAP Error Codes and Descriptions

Value	Text (English version)	Detailed Description
LDAP_SUCCESS	Success	The request was successful.
LDAP_OPERATIONS_ERROR	Operations error	An operations error occurred.
LDAP_PROTOCOL_ERROR	Protocol error	A protocol violation was detected.
LDAP_TIMELIMIT_EXCEEDED	Timelimit exceeded	An LDAP time limit was exceeded.
LDAP_SIZELIMIT_EXCEEDED	Sizelimit exceeded	An LDAP size limit was exceeded.
LDAP_COMPARE_FALSE	Compare false	A compare operation returned false.
LDAP_COMPARE_TRUE	Compare true	A compare operation returned true.
LDAP_STRONG_AUTH_NOT_SUPPORTED	Strong authentication not supported	The LDAP server does not support strong authentication.
LDAP_STRONG_AUTH_REQUIRED	Strong authentication required	Strong authentication is required for the operation.
LDAP_PARTIAL_RESULTS	Partial results and referral received	Partial results only returned.
LDAP_REFERRAL	Referral returned	Referral returned.
LDAP_ADMIN_LIMIT_EXCEEDED	Administration limit exceeded	Administration limit exceeded.
LDAP_UNAVAILABLE_CRITICAL_EXTENSION	Critical extension not supported	Critical extension is not supported.
LDAP_CONFIDENTIALITY_REQUIRED	Confidentiality is required	Confidentiality is required.
LDAP_SASLBIND_IN_PROGRESS	SASL bind in progress	An SASL bind is in progress.
LDAP_NO_SUCH_ATTRIBUTE	No such attribute	The attribute type specified does not exist in the entry.
LDAP_UNDEFINED_TYPE	Undefined attribute type	The attribute type specified is not valid.
LDAP_INAPPROPRIATE_MATCHING	Inappropriate matching	Filter type not supported for the specified attribute.

Table 2. LDAP Error Codes and Descriptions (continued)

Value	Text (English version)	Detailed Description
LDAP_CONSTRAINT_VIOLATION	Constraint violation	An attribute value specified violates some constraint (for example, a postal Address has too many lines, or a line that is too long).
LDAP_TYPE_OR_VALUE_EXISTS	Type or value exists	An attribute type or attribute value specified already exists in the entry.
LDAP_INVALID_SYNTAX	Invalid syntax	An attribute value that is not valid was specified.
LDAP_NO_SUCH_OBJECT	No such object	The specified object does not exist in the directory.
LDAP_ALIAS_PROBLEM	Alias problem	An alias in the directory points to a nonexistent entry.
LDAP_INVALID_DN_SYNTAX	Invalid DN syntax	A DN that is syntactically not valid was specified.
LDAP_IS_LEAF	Object is a leaf	The object specified is a leaf.
LDAP_ALIAS_DEREF_PROBLEM	Alias dereferencing problem	A problem was encountered when dereferencing an alias.
LDAP_INAPPROPRIATE_AUTH	Inappropriate authentication	Inappropriate authentication was specified (for example, LDAP_AUTH_SIMPLE was specified and the entry does not have a user Password attribute).
LDAP_INVALID_CREDENTIALS	Invalid credentials	Invalid credentials were presented (for example, the wrong password).
LDAP_INSUFFICIENT_ACCESS	Insufficient access	The user has insufficient access to perform the operation.
LDAP_BUSY	DSA is busy	The DSA is busy.
LDAP_UNAVAILABLE	DSA is unavailable	The DSA is unavailable.
LDAP_UNWILLING_TO_PERFORM	DSA is unwilling to perform	The DSA is unwilling to perform the operation.
LDAP_LOOP_DETECT	Loop detected	A loop was detected.
LDAP_NAMING_VIOLATION	Naming violation	A naming violation occurred.
LDAP_OBJECT_CLASS_VIOLATION	Object class violation	An object class violation occurred (for example a ″required″ attribute was missing from the entry).
LDAP_NOT_ALLOWED_ON_NONLEAF	Operation not allowed on nonleaf	The operation is not allowed on a nonleaf object.
LDAP_NOT_ALLOWED_ON_RDN	Operation not allowed on RDN	The operation is not allowed on an RDN.
LDAP_ALREADY_EXISTS	Already exists	The entry already exists.
LDAP_NO_OBJECT_CLASS_MODS	Cannot modify object class	Object class modifications are not allowed.
LDAP_RESULTS_TOO_LARGE	Results too large	Results too large.
LDAP_AFFECTS_MULTIPLE_DSAS	Affects multiple DSAs	Affects multiple DSAs.
LDAP_OTHER	Unknown error	An unknown error occurred.
LDAP_SERVER_DOWN	Can’t contact LDAP server	The LDAP library cannot contact the LDAP server.
LDAP_LOCAL_ERROR	Local error	Some local error occurred. This is usually a failed memory allocation.
LDAP_ENCODING_ERROR	Encoding error	An error was encountered encoding parameters to send to the LDAP server.

Table 2. LDAP Error Codes and Descriptions (continued)

Value	Text (English version)	Detailed Description
LDAP_DECODING_ERROR	Decoding error	An error was encountered decoding a result from the LDAP server.
LDAP_TIMEOUT	Timed out	A timelimit was exceeded while waiting for a result.
LDAP_AUTH_UNKNOWN	Unknown authentication method	The authentication method specified on a bind operation is not known.
LDAP_FILTER_ERROR	Bad search filter	An invalid filter was supplied to ldap_search (for example, unbalanced parentheses).
LDAP_USER_CANCELLED	User cancelled operation	The user cancelled the operation.
LDAP_PARAM_ERROR	Bad parameter to an ldap routine	An ldap routine was called with a bad parameter (for example, a NULL ld pointer, etc.).
LDAP_NO_MEMORY	Out of memory	A memory allocation (for example, malloc) call failed in an LDAP library routine.
LDAP_CONNECT_ERROR	Connection error	Connection error.
LDAP_NOT_SUPPORTED	Not supported	Not supported.
LDAP_CONTROL_NOT_FOUND	Control not found	Control not found.
LDAP_NO_RESULTS_RETURNED	No results returned	No results returned.
LDAP_MORE_RESULTS_TO_RETURN	More results to return	More results to return
LDAP_URL_ERR_NOTLDAP	URL doesn't begin with ldap://	The URL does not begin with ldap://
LDAP_URL_ERR_NODN	URL has no DN (required)	The URL does not have a DN (required).
LDAP_URL_ERR_BADSCOPE	URL scope string is invalid	The URL scope string is not valid.
LDAP_URL_ERR_MEM	Can't allocate memory space	Cannot allocate memory space.
LDAP_CLIENT_LOOP	Client loop	Client loop.
LDAP_REFERRAL_LIMIT_EXCEEDED	Referral limit exceeded	Referral limit exceeded.
LDAP_SSL_ALREADY_INITIALIZED	ldap_ssl_client_init successfully called previously in this process	The ldap_ssl_client_init was successfully called previously in this process.
LDAP_SSL_INITIALIZE_FAILED	Initialization call failed	SSL Initialization call failed.
LDAP_SSL_CLIENT_INIT_NOT_CALLED	Must call ldap_ssl_client_init before attempting to use SSL connection	Must call ldap_ssl_client_init before attempting to use SSL connection.
LDAP_SSL_PARAM_ERROR	Invalid SSL parameter previously specified	An SSL parameter that was not valid was previously specified.
LDAP_SSL_HANDSHAKE_FAILED	Failed to connect to SSL server	Failed to connect to SSL server.

Related Topics

ldap_memfree
ldap_parse_result

ldap_first_attribute

ldap_count_attributes
ldap_first_attribute
ldap_next_attribute

Purpose

Step through LDAP entry attributes.

Format

```
#include <ldap.h>

int ldap_count_attributes(
        LDAP *ld,
        LDAPMessage *entry );


char *ldap_first_attribute(
        LDAP *ld,
        LDAPMessage *entry,
        BerElement **ber)


char *ldap_next_attribute(
        LDAP *ld,
        LDAPMessage *entry,
        BerElement *ber)
```

Parameters

Input

ld Specifies the LDAP handle returned by a previous call to **ldap_open**, **ldap_ssl_init**, or **ldap_init**.

entry
The attribute information as returned by **ldap_first_entry** or **ldap_next_entry**.

Output

ber
Returns a pointer to a BerElement structure that is allocated to keep track of its current position.

Usage

Given an LDAP handle and an LDAPMessage, the **ldap_count_attributes** API returns the number of attributes contained in the returned entry. In many cases, it is desirable to know the total number of attributes contained in an LDAPMessage that was returned from an LDAP search operation.

The **ldap_count_attributes** API is designed to accept a pointer to the LDAPMessage structure returned from calls to **ldap_first_entry** and **ldap_next_entry**.

The **ldap_first_attribute** and **ldap_next_attribute** APIs are used to step through the attributes in an LDAP entry. The **ldap_first_attribute** API takes an *entry* as returned by **ldap_first_entry** or **ldap_next_entry** and returns a pointer to a buffer containing the name of the first attribute type in the entry. This buffer must be deallocated when its use is completed using **ldap_memfree**.

The pointer returned in *ber* should be passed to subsequent calls to **ldap_next_attribute** and is used to step through the entry's attributes. This pointer is deallocated by **ldap_next_attribute** when there are no more attributes (that is, when **ldap_next_attribute** returns **NULL**). Otherwise, the caller is responsible for deallocating the BerElement pointed to by *ber* when it is no longer needed by calling **ldap_memfree**.

The attribute names returned by **ldap_first_attribute** and **ldap_next_attribute** are suitable for inclusion in a call to **ldap_get_values** or **ldap_get_values_len** to retrieve the attribute's values. Following is an example:

```
for (attrtype=ldap_first_attribute (ld, entry, &ber);
        attrtype != NULL;
        attrtype=ldap_next_attribute (ld, entry, ber)) {
        /* calls to ldap_get_values or ldap_get_values_len
         * to parse the attribute values
         */
        ldap_memfree (attrtype);
{
```

The **ldap_next_attribute** API returns a string that contains the name of the next type in the entry. This string must be deallocated using **ldap_memfree** when its use is completed.

The *ber* parameter, as returned by **ldap_next_attribute**, is a pointer to a BerElement structure that was allocated by **ldap_first_attribute** to keep track of the current position in the LDAP result. This pointer is passed to **ldap_next_attribute** and is used to step through the entry's attributes. This pointer is deallocated by **ldap_next_attribute** when there are no more attributes (that is, when **ldap_next_attribute** returns **NULL**). Otherwise, the caller is responsible for deallocating the BerElement structure pointed to by *ber* when it is no longer needed by calling **ldap_memfree**.

Error Conditions

If an error occurs for **ldap_first_attribute** and **ldap_next_attribute**, **NULL** is returned. Use **ldap_get_errno** to retrieve the error value. See "ldap_error" on page 36 for possible values.

The **ldap_count_attributes** API returns -1 in case of an error. Use **ldap_get_errno** to retrieve the error value. See "ldap_error" on page 36 for possible values.

Related Topics

ldap_first_entry/reference
ldap_memfree
ldap_error
ldap_get_values

ldap_first_entry/reference

ldap_first_entry
ldap_next_entry
ldap_first_reference
ldap_next_reference
ldap_count_entries
ldap_count_references
ldap_get_entry_controls_np
ldap_parse_reference_np

Purpose

LDAP result entry and continuation reference parsing and counting APIs.

Format

```
#include <ldap.h>

LDAPMessage *ldap_first_entry(
        LDAP *ld,
        LDAPMessage *result)


LDAPMessage *ldap_next_entry(
        LDAP *ld,
        LDAPMessage *entry)


LDAPMessage *ldap_first_reference(
        LDAP *ld,
        LDAPMessage *result)


LDAPMessage *ldap_next_reference(
        LDAP *ld,
        LDAPMessage *ref,
        LDAPMessage *entry)


int ldap_count_entries(
        LDAP *ld,
        LDAPMessage *result)


int ldap_count_references(
        LDAP *ld,
        LDAPMessage *result)


int ldap_get_entry_controls_np(
        LDAP *ld,
        LDAPMessage *entry
        LDAPControl ***serverctrlsp)


int ldap_parse_reference_np(
        LDAP *ld,
        LDAPMessage *ref,
        char ***referralsp,
        LDAPControl ***serverctrlsp,
        int freeit)
```

Parameters

Input

ld Specifies the LDAP handle returned by a previous call to **ldap_open**, **ldap_ssl_init**, or **ldap_init**.

entry
Specifies a pointer to an entry returned on a previous call to **ldap_first_entry** or **ldap_next_entry**.

result
Specifies the result as returned by a call to **ldap_result** or to one the synchronous LDAP search routines (see “ldap_search” on page 73).

serverctrlsp
Specifies a pointer to a result parameter that is filled in with an allocated array of controls copied out of the **LDAPMessage** message. The control array should be freed by calling **ldap_controls_free**.

ref Specifies a pointer to a search continuation reference returned on a previous call to **ldap_first_reference** or **ldap_next_reference**.

referralsp
Specifies a pointer to a result parameter that is filled in with the contents of the referrals field from the **LDAPMessage** message, indicating zero or more alternate LDAP servers where the request should be retried. The referrals array should be freed by calling **ldap_value_free**. **NULL** may be supplied for this parameter to ignore the referrals field.

freeit
Specifies a boolean value that determines if the LDAP result chain (as specified by *ref*) is to be freed. Any nonzero value will result in the LDAP result chain being freed after the requested information is extracted. Alternatively, the **ldap_msgfree** API can be used to free the LDAP result chain at a later time.

Usage

These APIs are used to parse results received from **ldap_result** or the synchronous LDAP search operation APIs.

Processing Entries

The **ldap_first_entry** and **ldap_next_entry** APIs are used to step through and retrieve the list of entries from a search result chain. When an LDAP operation completes and the result is obtained as described, a list of LDAPMessage structures is returned. This is referred to as the search result chain. A pointer to the first of these structures is returned by **ldap_result** and **ldap_search_s**.

The **ldap_first_entry** API parses results received from **ldap_result** or the synchronous LDAP search operation routines and returns a pointer to the first entry in the result. If no entries were present in the result, NULL is returned. This pointer should be supplied on a subsequent call to **ldap_next_entry** to get the next entry, and so on until **ldap_next_entry** returns NULL. The **ldap_next_entry** API returns NULL when there are no more entries.

The **ldap_next_entry** API is used to parse results received from **ldap_result** or the synchronous LDAP search operation routines. The **ldap_next_entry** API returns **NULL** when there are no more entries.

The entry returned from **ldap_first_entry** and **ldap_next_entry** is used in calls to other parsing routines, such as **ldap_get_dn** and **ldap_first_attribute**. Following is an example:

```
for (entry=ldap_first_entry (ld, result);
        entry != NULL;
        entry=ldap_next_entry (ld, entry)) {
        /* calls to ldap_get_dn or ldap_first_attribute and
         * other routines to use the entry
         */
{
```

ldap_first_entry/reference

The **ldap_get_entry_controls_np** API is used to retrieve an array of server controls returned in an individual entry in a chain of search results.

Processing Continuation References

The **ldap_first_reference** and **ldap_next_reference** APIs are used to step through and retrieve the list of continuation references from a search result chain. They will return NULL when no more continuation references exist in the result set to be returned.

The **ldap_first_reference** API is used to retrieve the first continuation reference in a chain of search results. It takes the result as returned by a call to **ldap_result** or **ldap_search_s**, **ldap_search_st**, or **ldap_search_ext_s** and returns a pointer to the continuation reference in the result.

The pointer returned from **ldap_first_reference** should be supplied on a subsequent call to **ldap_next_reference** to get the next continuation reference.

The **ldap_parse_reference_np** API is used to retrieve the list of alternate servers returned in an individual continuation reference in a chain of search results. This API is also used to obtain an array of server controls returned in the continuation reference.

Counting Entries and References

The **ldap_count_entries** API is used to parse results received from **ldap_result** or the synchronous LDAP search operation routines in order to count the number of entries in the result. The number of entries in the chain of search results is returned. It can also be used to count the number of entries that remain in a chain if called with a message, entry, or continuation reference returned by **ldap_first_message**, **ldap_next_message**, **ldap_first_entry**, **ldap_next_entry**, **ldap_first_reference**, or **ldap_next_reference**, respectively.

The **ldap_count_references** API is used to count the number of continuation references returned. It can also be used to count the number of continuation references that remain in a chain.

Error Conditions

If an error occurs in **ldap_first_entry**, **ldap_next_entry**, **ldap_first_reference**, or **ldap_next_reference**, **NULL** is returned. Use **ldap_get_errno** to retrieve the error value. See “ldap_error” on page 36 for possible values.

The **ldap_count_entries** or **ldap_count_references** APIs return -1 in case of error. Use **ldap_get_errno** to retrieve the error value. See “ldap_error” on page 36 for possible values.

The **ldap_get_entry_controls_np** and **ldap_parse_reference_np** APIs return **LDAP_SUCCESS** if successful, otherwise an error code is returned. See “ldap_error” on page 36 for possible values.

Related Topics

ldap_result
ldap_first_attribute
ldap_get_dn
ldap_search
ldap_get_values

ldap_get_dn

ldap_get_dn
ldap_explode_dn

Purpose

LDAP DN handling routines.

Format

```
#include <ldap.h>

char *ldap_get_dn(
        LDAP *ld,
        LDAPMessage *entry)


char **ldap_explode_dn(
        char *dn,
        int notypes)
```

Parameters

Input

ld Specifies the LDAP handle returned by a previous call to **ldap_open**, **ldap_ssl_init**, or **ldap_init**.

entry
Specifies attribute information as returned by **ldap_first_entry** or **ldap_next_entry**.

dn Specifies the distinguished name of the entry to be parsed.

notypes
Requests that only the relative distinguished name (RDN) values be returned, not their types. For example, the DN `cn=Bob, c=US` would return as either `{"cn=Bob", "c=US", NULL}` or `{"Bob", "US", NULL}`. depending on whether *notypes* was 0 or 1, respectively.

Usage

The **ldap_get_dn** API takes an *entry* as returned by **ldap_first_entry** or **ldap_next_entry**, and returns a copy of the entry's DN. Space for the DN is obtained on the caller's behalf and should be deallocated by the caller using **ldap_memfree**.

The **ldap_explode_dn** API takes a DN as returned by **ldap_get_dn** and breaks it up into its component parts. Each part is known as a relative distinguished name (RDN). The **ldap_explode_dn** API returns a NULL-terminated array of character strings, each component of which contains an RDN from the DN. This routine allocates memory that the caller must deallocate using **ldap_value_free**.

Error Conditions

If an error occurs, **NULL** is returned. For the **ldap_get_dn** API, use **ldap_get_errno** to retrieve the error value. See "ldap_error" on page 36 for possible values. For the **ldap_explode_dn** API, specific error information is not available using **ldap_get_errno**. Possible errors are: NULL pointer passed into the function, memory allocation error, or the string passed in was not parsable as a distinguished name.

Related Topics

ldap_error
ldap_first_entry
ldap_value_free

ldap_get_values

ldap_get_values
ldap_get_values_len
ldap_count_values
ldap_count_values_len
ldap_value_free
ldap_value_free_len

Purpose

LDAP attribute value handling APIs.

Format

```
#include <ldap.h>

typedef struct berval {
    unsigned long bv_len;
    char *bv_val;
};


char **ldap_get_values(
        LDAP *ld,
        LDAPMessage *entry,
        char *attr)


struct berval **ldap_get_values_len(
        LDAP *ld,
        LDAPMessage *entry,
        char *attr)


int ldap_count_values(
        char **vals)


int ldap_count_values_len(
        struct berval **bvals)


void ldap_value_free(
        char **vals)


void ldap_value_free_len(
        struct berval **bvals)
```

Parameters

Input

ld Specifies the LDAP handle returned by a previous call to **ldap_open**, **ldap_ssl_init**, or **ldap_init**.

entry
Specifies the LDAP entry from which to retrieve the attribute values.

attr
Specifies the attribute type to retrieve. It may be an attribute type as returned from **ldap_first_attribute** or **ldap_next_attribute**, or if the attribute type is known it can simply be given.

vals
Specifies a pointer to a NULL-terminated array of attribute values returned by **ldap_get_values**.

bvals
Specifies a pointer to a NULL-terminated array of pointers to berval structures, as returned by **ldap_get_values_len**.

Usage

These APIs retrieve and manipulate attribute values from an LDAP entry as returned by **ldap_first_entry** or **ldap_next_entry**. The result of **ldap_get_values** is a NULL-terminated array of NULL-terminated character strings that represent the attributes values. The **ldap_get_values** API converts the returned results into a NULL-terminated string in the codeset of the current locale. The data is assumed to be (UTF-8) coming from the LDAP server. If the data is binary data or conversions should be avoided then the **ldap_get_values_len** API must be used.

The **ldap_get_values** API allocates memory that the caller must deallocate using **ldap_value_free**.

Use the **ldap_get_values_len** API if the attribute values are binary in nature and not suitable to be returned as an array of NULL-terminated character strings. The **ldap_get_values_len** API returns a NULL-terminated array of pointers to berval structures, each containing the length of and a pointer to a value.

The **ldap_get_values_len** API allocates memory that the caller must deallocate using **ldap_value_free_len**.

The **ldap_count_values** API counts values in an array of attribute values as returned by **ldap_get_values**. The number of attribute values is returned.

The **ldap_count_values_len** API counts the number of values in a NULL-terminated array of pointers to berval structures where each represents an attribute value. The number of attribute values is returned.

The **ldap_value_free** API deallocates an array of attribute values that was allocated by **ldap_get_values**. Following is an example of its usage:

```
for (attrtype=ldap_first_attribute (ld, entry, &ber);
        attrtype != NULL;
        attrtype=ldap_next_attribute (ld, entry, ber)) {
        char *values[];
        values=ldap_get_values (ld, entry, attrtype);
        /*
         *  work with the attribute type and values
         */
        ldap_value_free(values);
}       ldap_memfree(attrtype);
```

The **ldap_value_free_len** API deallocates an array of attribute values that was allocated by **ldap_get_values_len**. Following is an example of its usage:

```
for (attrtype=ldap_first_attribute (ld, entry, &ber);
        attrtype != NULL;
        attrtype=ldap_next_attribute (ld, entry, ber)) {
        struct berval *bvals[];
        bvals=ldap_get_values_len (ld, entry, attrtype);
        /*
         *  work with the attribute type and values
         */
        ldap_value_free_len(bvals);
        ldap_memfree(attrtype);
}
```

Error Conditions

If no values are found or an error occurs in **ldap_get_values** or **ldap_get_values_len**, **NULL** is returned. Use **ldap_get_errno** to retrieve the error value. See “ldap_error” on page 36 for possible values.

ldap_get_values

Related Topics

ldap_first_entry/reference
ldap_first_attribute
ldap_error

ldap_init

ldap_init
ldap_open (deprecated)
ldap_set_option
ldap_set_option_np (nonportable)
ldap_get_option

Purpose

Initialize the LDAP library, open a connection to an LDAP server, and get or set options for an LDAP connection.

If you want to use the socksified client, see “Using the Socksified Client” on page 5.

Format

```
#include <ldap.h>

LDAP *ldap_init(
        char *host,
        int port)


LDAP *ldap_open(
        char *host,
        int port)


int ldap_set_option(
        LDAP *ld,
        int optionToSet,
        void *optionValue)


int ldap_set_option_np(
        LDAP *ld,
        int optionToSet,
        optionValue)


int ldap_get_option(
        LDAP *ld,
        int optionToGet,
        void *optionValue)
```

Parameters

Input

host
Specifies the name of the host on which the LDAP server is running. It can contain a space-separated list of hosts in which to try to connect, and each host may optionally be of the form *host:port*. If present, *:port* overrides the *port* parameter to **ldap_init** or **ldap_open**. Following are some examples:

```
myhost.mycompany.com
myhost.mycompany.com:389 yourhost.yourcompany.com
```

If *host* is NULL, the LDAP server is assumed to be running on the local host.

port
Specifies the TCP/IP port number in which to connect. If the default IANA-assigned port of 389 is desired, **LDAP_PORT** should be specified. To use the default SSL port 636 for SSL connections, use **LDAPS_PORT**.

ldap_init

ld Specifies the LDAP handle returned by a previous call to **ldap_open**, **ldap_ssl_init**, or **ldap_init**.

optionToSet
Specifies which LDAP option's value should be set. See "Setting and Getting Session Settings" on page 51 for the list of supported options.

optionToGet
Specifies which LDAP option's value should be returned. See "Setting and Getting Session Settings" on page 51 for the list of supported options.

optionValue
Depending on the operation, protocol version, or both, *optionValue* specifies the value, or address of the value, to be set through **ldap_set_option** or **ldap_set_option_np**. For **ldap_get_option**, it specifies the address of the storage in which to return the queried value. The following table details the format of the *optionValue* parameter to be specified.

Table 3. The optionValue Parameter Specifications

optionToSet **or** *optionToGet*	**ldap_set_option (Version 3)**	**ldap_set_option (Version 2)**	**ldap_set_option_np**	**ldap_get_option**
LDAP_OPT_SIZELIMIT	int *	int	int	int *
LDAP_OPT_TIMELIMIT	int *	int	int	int *
LDAP_OPT_REFHOPLIMIT	int *	int	int	int *
LDAP_OPT_DEREF	int *	int	int	int *
LDAP_OPT_RESTART	int (ON/OFF)	int (ON/OFF)	int (ON/OFF)	int *
LDAP_OPT_REFFERALS	int (ON/OFF)	int (ON/OFF)	int (ON/OFF)	int *
LDAP_OPT_DEBUG	int *	int	int	int *
LDAP_OPT_SSL_CIPHER	char *	char *	char *	char **
LDAP_OPT_SSL_TIMEOUT	int *	int	int	int *
LDAP_OPT_REBIND_FN	LDAPRebindProc *	LDAPRebindProc *	LDAPRebindProc *	LDAPRebindProc **
LDAP_OPT_PROTOCOL_VERSION	int *	int *	int	int *
LDAP_OPT_SERVER_CONTROLS	LDAPControl **	N/A	LDAPControl **	LDAPControl ***
LDAP_OPT_CLIENT_CONTROLS	LDAPControl **	N/A	LDAPControl **	LDAPControl ***
LDAP_OPT_UTF8_IO	int (ON/OFF)	int (ON/OFF)	int (ON/OFF)	int *
LDAP_OPT_V2_WIRE_FORMAT	int	int	int	int *
LDAP_OPT_HOST_NAME	n/a	n/a	n/a	char *
LDAP_OPT_ERROR_NUMBER	n/a	n/a	n/a	int *
LDAP_OPT_ERROR_STRING	n/a	n/a	n/a	char *
LDAP_OPT_EXT_ERROR	n/a	n/a	n/a	int *
Note: The ON and OFF in the table refer to **LDAP_OPT_ON** and **LDAP_OPT_OFF**, respectively.				

Usage

The **ldap_init** API initializes a session with an LDAP server. The server is not actually contacted until an operation is preformed that requires it, allowing various options to be set after initialization, but before actually contacting the host. It allocates an LDAP handle which is used to identify the connection and maintain per-connection information.

For SSL, the equivalent of ldap_init is **ldap_ssl_init**. The **ldap_ssl_init** API is used to initialize a secure SSL session with a server. See "ldap_ssl" on page 77 for more information.

Although still supported, the use of **ldap_open** is deprecated. The **ldap_open** API allocates an LDAP handle and opens a connection to the LDAP server. Use of **ldap_init** instead of **ldap_open** is recommended.

For **ldap_open**, the **ldap_ssl_start** API starts a secure (SSL) connection to an LDAP server.

The **ldap_init** and **ldap_open** APIs return a handle that is passed to subsequent calls to **ldap_bind**, **ldap_search**, and so on.

The **ldap_set_option** and **ldap_set_option_np** APIs modify the current value of an option used by the LDAP programming interface. These options take on default values after **ldap_open** or **ldap_init** is called and their current value can be retrieved using the **ldap_get_option** API. On successful completion, the current value of the requested option is set to the value specified by the *optionValue* parameter with the return code set to **LDAP_SUCCESS**.

Environmental Variables Affecting Session Settings

There are three environment variables that can affect the session settings. One, **LDAP_DEBUG**, is discussed in "Tracing" on page 14. Setting the **LDAP_DEBUG** environment variable has the same effect as calling **ldap_set_option** to set the **LDAP_OPT_DEBUG** session option.

The **LDAP_VERSION** environment variable can be used to establish the LDAP version to be used for a session. Setting the **LDAP_VERSION** environment variable has the same effect as calling **ldap_set_option** to set the **LDAP_OPT_PROTOCOL_VERSION** session option. Valid values for the **LDAP_VERSION** environment variable are **2** and **3**. See "LDAP_OPT_PROTOCOL_VERSION" on page 56 for more information.

The **LDAP_V2_WIRE_FORMAT** environment variable can be used to establish the wire format to be used for Version 2 data exchanged between the client library APIs and the target LDAP server. Setting the **LDAP_V2_WIRE_FORMAT** environment variable has the same effect as calling **ldap_set_option** to set the **LDAP_OPT_V2_WIRE_FORMAT** session option. Valid values for the **LDAP_V2_WIRE_FORMAT** environment variable are **UTF8** and **ISO8859-1**. See "LDAP_OPT_V2_WIRE_FORMAT" on page 57 for more information.

Setting and Getting Session Settings

The **ldap_get_option**, **ldap_set_option**, and **ldap_set_option_np** APIs can be used to:

- Get or set the maximum number of entries that can be returned on a search operation. (**LDAP_OPT_SIZELIMIT**)
- Get or set the maximum number of seconds to wait for search results. (**LDAP_OPT_TIMELIMIT**)
- Get or set the maximum number of referrals in a sequence that the client can follow. (**LDAP_OPT_REFHOPLIMIT**)
- Get or set the rules for following aliases at the server. (**LDAP_OPT_DEREF**)
- Get or set whether select system call should be restarted. (**LDAP_OPT_RESTART**)
- Get or set whether referrals should be followed by the client. (**LDAP_OPT_REFERRALS**)
- Get or set the debug options. (**LDAP_OPT_DEBUG**)
- Get or set the SSL ciphers to use. (**LDAP_OPT_SSL_CIPHER**)
- Get or set the SSL time-out for refreshing session keys. (**LDAP_OPT_SSL_TIMEOUT**)
- Get or set the address of application's rebind procedure. (**LDAP_OPT_REBIND_FN**)
- Get or set the LDAP protocol version to use (Version 2 or Version 3). (**LDAP_OPT_PROTOCOL_VERSION**)
- Get or set the default server controls. (**LDAP_OPT_SERVER_CONTROLS**)
- Get or set the default client library controls. (**LDAP_OPT_CLIENT_CONTROLS**)
- Get or set the format of textual data. (**LDAP_OPT_UTF8_IO**)
- Get or set the format of textual data when using V2 protocol. (**LDAP_OPT_V2_WIRE_FORMAT**)
- Get the current host name (cannot be set). (**LDAP_OPT_HOST_NAME**)

ldap_init

- Get the error number (cannot be set). (**LDAP_OPT_ERROR_NUMBER**)
- Get the error string (cannot be set). (**LDAP_OPT_ERROR_STRING**)

If your LDAP application is based on the LDAP Version 2 APIs and uses the **ldap_set_option** or **ldap_get_option** functions (that is, you are using **ldap_open** or your application uses **ldap_init** and **ldap_set_option** to switch from the default of LDAP Version 3 to use the LDAP Version 2 protocol and subsequently uses the **ldap_set_option** or **ldap_get_option** calls), see “ldap_set_option Syntax for LDAP Version 2 Applications” on page 58 for important information.

For a description of the differences between the **ldap_set_option** API and the **ldap_set_option_np** (nonportable) API, see “Comparing the ldap_set_option and ldap_set_option_np APIs” on page 58.

Additional details on specific options for **ldap_get_option**, **ldap_set_option**, and **ldap_set_option_np** are provided in the following sections.

LDAP_OPT_SIZELIMIT

Specifies the maximum number of entries that can be returned on a search operation.

Note: The actual size limit for operations is also bounded by the maximum number of entries that the server is configured to return. Thus, the actual size limit will be the lesser of the value specified on this option and the value configured in the LDAP server. The default size limit is unlimited, specified with a value of zero (thus deferring to the size limit setting of the LDAP server). A value of zero (the default) means no limit

Examples:

```
int sizevalue=50;
ldap_set_option(ld, LDAP_OPT_SIZELIMIT, (void *) &sizevalue); /*Version 3 protocol*/
    or
ldap_set_option(ld, LDAP_OPT_SIZELIMIT, (void *) sizevalue); /*Version 2 protocol*/
    or
ldap_set_option_np(ld, LDAP_OPT_SIZELIMIT, (int) sizevalue);

ldap_get_option(ld, LDAP_OPT_SIZELIMIT, (void *) &sizevalue);
```

LDAP_OPT_TIMELIMIT

Specifies the number of seconds to wait for search results. Note that the actual time limit for operations is also bounded by the maximum time that the server is configured to allow. Thus, the actual time limit will be the lesser of the value specified on this option and the value configured in the LDAP server. The default is unlimited (specified with a value of zero).

Examples:

```
int timevalue=50;
ldap_set_option(ld, LDAP_OPT_TIMELIMIT, (void *) &timevalue); /*Version 3 protocol*/
    or
ldap_set_option(ld, LDAP_OPT_TIMELIMIT, (void *) timevalue); /*Version 2 protocol*/
    or
ldap_set_option_np(ld, LDAP_OPT_TIMELIMIT, (int) timevalue);

ldap_get_option(ld, LDAP_OPT_TIMELIMIT, (void *) &timevalue);
```

LDAP_OPT_REHOPLIMIT

Specifies the maximum number of servers to contact when chasing referrals. For subtree searches, this is the limit on the depth of nested search references, so the number of servers contacted might actually exceed this value. The default is 10.

Examples:

```
int hoplimit=7;
ldap_set_option( ld, LDAP_OPT_REFHOPLIMIT, (void *) &hoplimit); /* Version 3 protocol */
    or
ldap_set_option( ld, LDAP_OPT_REFHOPLIMIT, (void *) hoplimit); /* Version 2 protocol */
    or
ldap_set_option_np( ld, LDAP_OPT_REFHOPLIMIT, (int) hoplimit);

ldap_get_option( ld, LDAP_OPT_REFHOPLIMIT, (void *) &hoplimit);
```

LDAP_OPT_DEREF

Specifies alternative rules for following aliases at the server. The default is **LDAP_DEREF_NEVER**.

Supported values:
- **LDAP_DEREF_NEVER** 0 (default)
- **LDAP_DEREF_SEARCHING** 1
- **LDAP_DEREF_FINDING** 2
- **LDAP_DEREF_ALWAYS** 3

The **LDAP_DEREF_FINDING** value means aliases should be dereferenced when locating the base object, but not during a search.

Examples:

```
int deref = LDAP_DEREF_NEVER;
ldap_set_option( ld, LDAP_OPT_DEREF, (void *) &deref); /* Version 3 protocol */
    or
ldap_set_option( ld, LDAP_OPT_DEREF, (void *) deref); /* Version 2 protocol */
    or
ldap_set_option_np( ld, LDAP_OPT_DEREF, (int) deref);

ldap_get_option( ld, LDAP_OPT_DEREF, (void *) &value);
```

LDAP_OPT_RESTART

Specifies whether the **select** system call should be restarted when it is interrupted by the system. The returned value will be one of **LDAP_OPT_ON** or **LDAP_OPT_OFF** (default).

Examples:

```
int value;
ldap_set_option( ld, LDAP_OPT_RESTART, (void *) LDAP_OPT_ON); /* Version 2 or 3 protocol */
    or
ldap_set_option_np( ld, LDAP_OPT_RESTART, (int) LDAP_OPT_ON);

ldap_get_option( ld, LDAP_OPT_RESTART, (void *) &value);
```

LDAP_OPT_REFERRALS

Specifies whether the LDAP library will automatically follow referrals returned by LDAP servers. It can be set to one of the constants **LDAP_OPT_ON** or **LDAP_OPT_OFF**. By default, the LDAP client will follow referrals.

Examples:

```
int value:
ldap_set_option( ld, LDAP_OPT_REFFERALS, (void *) LDAP_OPT_ON); /* Version 2 or 3 protocol */
    or
ldap_set_option_np( ld, LDAP_OPT_REFFERALS, (int) LDAP_OPT_ON);

ldap_get_option( ld, LDAP_OPT_REFFERALS, (void *) &value);
```

LDAP_OPT_DEBUG

Specifies a bit map that indicates the level of debug trace for the LDAP library. The *optionValue* parameter can be specified as either an integer greater than or equal to zero or as any bitwise ″ored″ (|) or ″added″ (+) combination of the identifiers:

- **LDAP_DEBUG_TRACE**
- **LDAP_DEBUG_PACKETS**
- **LDAP_DEBUG_ARGS**
- **LDAP_DEBUG_CONNS**
- **LDAP_DEBUG_BER**
- **LDAP_DEBUG_FILTER**
- **LDAP_DEBUG_CONFIG**
- **LDAP_DEBUG_ACL**
- **LDAP_DEBUG_STATS**
- **LDAP_DEBUG_STATS2**
- **LDAP_DEBUG_SHELL**
- **LDAP_DEBUG_PARSE**

In addition, **LDAP_DEBUG_OFF** or **LDAP_DEBUG_ANY** are accepted.

LDAP_OPT_DEBUG is a global option (it does not pertain to any particular LDAP handle), whereas the other options pertain to a specific LDAP handle. For example, you can set the search time limit to 10 seconds for one server using one LDAP handle, but you could allow it to default to 0 (no time limit) for a second server using a different LDAP handle. **LDAP_OPT_DEBUG** applies to all allocated LDAP handles.

Examples:

```
int debugvalue= LDAP_DEBUG_TRACE + LDAP_DEBUG_PACKETS;
ldap_set_option( ld, LDAP_OPT_DEBUG, (void *) &debugvalue); /* Version 3 protocol */
    or
ldap_set_option( ld, LDAP_OPT_DEBUG, (void *) debugvalue); /* Version 2 protocol */
    or
ldap_set_option_np( ld, LDAP_OPT_DEBUG, (int) debugvalue);

ldap_get_option( ld, LDAP_OPT_DEBUG, (void *) &debugvalue);
```

Example turning all traces on:

```
int debugvalue=LDAP_DEBUG_ANY;
ldap_set_option(ld, LDAP_OPT_DEBUG, (void *) &debugvalue); /* Version 3 protocol */
    or
ldap_set_option(ld, LDAP_OPT_DEBUG, (void *) LDAP_DEBUG_ANY); /* Version 2 protocol */
    or
ldap_set_option_np(ld, LDAP_OPT_DEBUG, (int) LDAP_DEBUG_ANY);
```

Example turning all tracing off:

```
int debugvalue=LDAP_DEBUG_OFF;
ldap_set_option(ld, LDAP_OPT_DEBUG, (void *) &debugvalue); /* Version 3 protocol */
    or
ldap_set_option(ld, LDAP_OPT_DEBUG, (void *) LDAP_DEBUG_OFF); /* Version 2 protocol */
    or
ldap_set_option_np(ld, LDAP_OPT_DEBUG, (int) LDAP_DEBUG_OFF);
```

Example tracing just BER encodings and functional flow tracepoints:

```
int debugvalue=LDAP_DEBUG_BER + LDAP_DEBUG_TRACE;
ldap_set_option(ld, LDAP_OPT_DEBUG, (void *) &debugvalue); /* Version 3 protocol */
    or
ldap_set_option(ld, LDAP_OPT_DEBUG, (void *) debugvalue); /* Version 2 protocol */
    or
ldap_set_option_np(ld, LDAP_OPT_DEBUG, (int) debugvalue);
```

Example tracing packets and connections:

```
int debugvalue=LDAP_DEBUG_PACKETS | LDAP_DEBUG_CONNS;
ldap_set_option(ld, LDAP_OPT_DEBUG, (void *) &debugvalue); /* Version 3 protocol */
    or
ldap_set_option(ld, LDAP_OPT_DEBUG, (void *) debugvalue); /* Version 2 protocol */
    or
ldap_set_option_np(ld, LDAP_OPT_DEBUG, (int) LDAP_DEBUG_PACKETS | LDAP_DEBUG_CONNS);
```

LDAP_OPT_SSL_CIPHER

Specifies a set of one or more ciphers to be used when negotiating the cipher algorithm with the LDAP server. The value for this option is specified as the *v3cipher_specs* value supplied to the **gsk_secure_soc_init** function call in System SSL. Refer to *z/OS: System Secure Sockets Layer Programming* for a description of supported cipher specifications and ordering their precedence. The cipher is a concatenation of a set of strings. As a convenience, the following strings are defined in **ldap.h**.

Supported ciphers:

- **LDAP_SSL_RC4_MD5_EX** ″03″
- **LDAP_SSL_RC2_MD5_EX** ″06″
- **LDAP_SSL_RC4_SHA_US** ″05″
- **LDAP_SSL_RC4_MD5_US** ″04″
- **LDAP_SSL_DES_SHA_US** ″09″
- **LDAP_SSL_3DES_SHA_US** ″0A″

Examples:

```
char *cipher = "090A";
char *cipher2 = LDAP_SSL_3DES_SHA_US LDAP_SSL_DES_SHA_US;
ldap_set_option( ld, LDAP_OPT_SSL_CIPHER, (void *) cipher); /* Version 2 or 3 protocol */

ldap_set_option_np( ld, LDAP_OPT_SSL_CIPHER, (char *) cipher2);

ldap_get_option( ld, LDAP_OPT_SSL_CIPHER, (void *) &cipher);
```

Note that **ldap_get_option** allocates storage for the returned cipher string. Use **ldap_memfree** to free this storage.

LDAP_OPT_SSL_TIMEOUT

Specifies in seconds the SSL inactivity timer. After the specified seconds, in which no SSL activity has occurred, the SSL connection will be refreshed with new session keys. A smaller value may help increase security, but will have an impact on performance. The default SSL time-out value is 43200 seconds.

Examples:

```
int value = 100;
ldap_set_option( ld, LDAP_OPT_SSL_TIMEOUT, (void *) &value); /* Version 3 protocol */
    or
ldap_set_option( ld, LDAP_OPT_SSL_TIMEOUT, (void *) value); /* Version 2 protocol */
    or
ldap_set_option_np( ld, LDAP_OPT_SSL_TIMEOUT, (int) value);

ldap_get_option( ld, LDAP_OPT_SSL_TIMEOUT, (void *) &value)
```

LDAP_OPT_REBIND_FN

Specifies the address of a routine to be called by the LDAP library when the need arises to authenticate a connection with another LDAP server. This can occur, for example, when the LDAP library is chasing a referral. If a routine is not defined, referrals will always be chased anonymously. A default routine is not defined.

ldap_init

Examples:

```
extern LDAPRebindProc proc_address;
LDAPRebindProc value;
ldap_set_option( ld, LDAP_OPT_REBIND_FN, (void *) &proc_address); /* Version 2 or 3 protocol */
    or
ldap_set_option_np( ld, LDAP_OPT_REBIND_FN, (LDAPRebindProc *) &proc_address);

ldap_get_option( ld, LDAP_OPT_REBIND_FN, (void *) &value);
```

LDAP_OPT_PROTOCOL_VERSION

Specifies the LDAP protocol to be used by the LDAP client library when connecting to an LDAP server. Also used to determine which LDAP protocol is being used for the connection. For an application that uses **ldap_init** to create the LDAP connection the default value of this option will be **LDAP_VERSION3** for communicating with the LDAP server. The default value of this option will be **LDAP_VERSION2** if the application uses the deprecated **ldap_open** API. In either case, the **LDAP_OPT_PROTOCOL_VERSION** option can be used with **ldap_set_option** to change the default. The LDAP protocol version should be reset prior to issuing the bind (or any operation that causes an implicit bind).

Examples:

```
version2 = LDAP_VERSION2;
version3 = LDAP_VERSION3;
int value;
/* Example for Version 3 application setting version to version 2 with ldap_set_option  */
ldap_set_option( ld, LDAP_OPT_PROTOCOL_VERSION, (void *) &version2);
/* Example of Version 2 application setting version to version 3 with ldap_set_option */
ldap_set_option( ld, LDAP_OPT_PROTOCOL_VERSION, (void *) &version3);
/* Example for Version 3 application setting version to version 2 with ldap_set_option_np  */
ldap_set_option_np( ld, LDAP_OPT_PROTOCOL_VERSION, (int) LDAP_VERSION2);
/* Example of Version 2 application setting version to version 3 with ldap_set_option_np */
ldap_set_option_np( ld, LDAP_OPT_PROTOCOL_VERSION, (int) LDAP_VERSION3);

ldap_get_option( ld, LDAP_OPT_PROTOCOL_VERSION, (void *) &value);
```

LDAP_OPT_SERVER_CONTROLS

Specifies a default list of server controls to be sent with each request. The default list can be overridden by specifying a server control, or list of server controls, on specific APIs. By default, there are no settings for server controls. Controls are only applicable when using the Version 3 LDAP protocol.

Example:

```
LDAPControl ** ctrlArray;
    .
    .
    .
ldap_set_option( ld, LDAP_OPT_SERVER_CONTROLS, (void *) &ctrlArray);
    or
ldap_set_option_np( ld, LDAP_OPT_SERVER_CONTROLS, (LDAPControl **) ctrlArray);

ldap_get_option( ld, LDAP_OPT_SERVER_CONTROLS, (void *) &ctrlArray);
```

Note that **ldap_get_option** returns a pointer to an array of LDAPControl structures. Use **ldap_controls_free** to free the storage allocated for this array.

LDAP_OPT_CLIENT_CONTROLS

Specifies a default list of client controls to be processed by the client library with each request. Since client controls are not defined for this version of the library, the **ldap_set_option** and **ldap_set_option_np** APIs can be used to define a set of default, noncritical client controls. If one or more client controls in the set is critical, the entire list is rejected with a return code of **LDAP_UNAVAILABLE_CRITICAL_EXTENSION**.

LDAP_OPT_UTF8_IO

Relative to the context LDAP handle, specifies the format of textual data exchanged (input/output) between the calling application and the LDAP client library APIs. **LDAP_OPT_ON** indicates textual I/O is in the UTF-8 codeset. **LDAP_OPT_OFF** indicates textual I/O is in the codeset of the current locale. **LDAP_OPT_OFF** is the default.

Note: This setting is only applicable to LDAP operations that accept an LDAP handle as input. Other LDAP operations (for example, **ldap_init**) require textual I/O to be in the codeset of the current locale.

Examples:

```
int value;
ldap_set_option( ld, LDAP_OPT_UTF8_IO, (void *) LDAP_OPT_ON );  /* Version 2 or 3 protocol */
    or
ldap_set_option_np( ld, LDAP_OPT_UTF8_IO, (int) LDAP_OPT_ON );

ldap_get_option( ld, LDAP_OPT_UTF8_IO, (void *) &value.);
```

LDAP_OPT_V2_WIRE_FORMAT

Relative to the context LDAP handle, specifies the format of textual data to be exchanged between the LDAP client library APIs and the LDAP server being contacted when using the Version 2 protocol. **LDAP_OPT_V2_WIRE_FORMAT_ISO8859_1** indicates that textual data is exchanged in ISO8859-1 format, which is the default for z/OS LDAP Version 2 servers. **LDAP_OPT_V2_WIRE_FORMAT_UTF8** indicates that textual data is exchanged in UTF-8 format, which is the default for z/OS LDAP Version 3 servers. Also note that many non-z/OS LDAP Version 3 servers expect to exchange data in UTF-8 format, regardless of the protocol version. **LDAP_OPT_V2_WIRE_FORMAT_UTF8** is the default in z/OS and OS/390 Release 8 and above.

Examples:

```
int value;
ldap_set_option(ld, LDAP_OPT_V2_WIRE_FORMAT, (void *) LDAP_OPT_V2_WIRE_FORMAT_ISO8859_1); /* V2 or V3 protocol */
  or
ldap_set_option_np(ld, LDAP_OPT_V2_WIRE_FORMAT, (int) LDAP_OPT_V2_WIRE_FORMAT_ISO8859_1);

ldap_get_option (ld, LDAP_OPT_V2_WIRE_FORMAT, &value);
```

LDAP_OPT_HOST_NAME

This is a read-only option that returns a pointer to the host name for the original connection (as specified on **ldap_init**, **ldap_ssl_init**, or **ldap_open**).

Example:

```
char * hostname;
ldap_get_option( ld, LDAP_OPT_HOST_NAME, (void *) &hostname);
```

Use **ldap_memfree** to free the memory allocated for the returned host name.

LDAP_OPT_ERROR_NUMBER

This is a read-only option that returns the error code associated with the most recent LDAP error that occurred for the specified LDAP connection.

Example:

```
int error;
ldap_get_option( ld, LDAP_OPT_ERROR_NUMBER, (void *) &error);
```

LDAP_OPT_ERROR_STRING

This is a read-only option that returns the text message associated with the most recent LDAP error that occurred for the specified LDAP connection.

Example:

```
char * error_string; ldap_get_option( ld, LDAP_OPT_ERROR_STRING, (void *) &error_string);
```

Use **ldap_memfree** to free the memory allocated for the returned error string.

LDAP_OPT_EXT_ERROR

This is a read-only option that returns the extended error code. For example, if an SSL error occurred when attempting to invoke an **ldap_search_s** API, the actual SSL error can be obtained by using **LDAP_OPT_EXT_ERROR**.

Example:

```
int exterror;
ldap_get_option( ld, LDAP_OPT_ERROR_EXTERROR, (void *) &exterror);
```

Returns errors reported by the SSL library.

Error Conditions

If an error occurs, the **ldap_init** and **ldap_open** APIs return **NULL**.

For **ldap_get_option**, and **ldap_set_option**, and **ldap_set_option_np**, **LDAP_PARM_ERROR** can be returned if the LDAP handle is not valid or if the requested option is not one of the accepted values.

ldap_set_option Syntax for LDAP Version 2 Applications

To maintain compatibility with older versions of the LDAP client library (before LDAP Version 3), the **ldap_set_option** API expects the value of the following option values to be supplied, instead of the address of the value, when the application is running as an LDAP Version 2 application:

- **LDAP_OPT_SIZELIMIT**
- **LDAP_OPT_TIMELIMIT**
- **LDAP_OPT_REFHOPLIMIT**
- **LDAP_OPT_SSL_TIMEOUT**
- **LDAP_OPT_DEREF**
- **LDAP_OPT_DEBUG**

The LDAP application is typically running as LDAP Version 2 when it uses **ldap_open** to create the LDAP connection. The LDAP application is typically running as LDAP Version 3 when it uses **ldap_init** to create the LDAP connection. Note that **LDAP_OPT_PROTOCOL_VERSION** can be used to toggle the protocol, in which case the behavior of **ldap_set_option** changes.

Comparing the ldap_set_option and ldap_set_option_np APIs

The **ldap_set_option** and **ldap_set_option_np** APIs support the same LDAP option value settings; they differ only in the level of indirection required to specify certain settings. The **ldap_set_option_np** API is a z/OS-specific API and its intent is to provide an alternate programming interface for setting LDAP option values. Furthermore, the rules for specifying values through **ldap_set_option_np** will not be subject to change in future releases. Unlike **ldap_set_option**, the **ldap_set_option_np** API expects the *value* of the following option values to be supplied, instead of the address of the value, regardless of the LDAP version setting:

- **LDAP_OPT_SIZELIMIT**
- **LDAP_OPT_TIMELIMIT**

- **LDAP_OPT_REFHOPLIMIT**
- **LDAP_OPT_SSL_TIMEOUT**
- **LDAP_OPT_PROTOCOL_VERSION**
- **LDAP_OPT_DEREF**
- **LDAP_OPT_DEBUG**

Related Topics

ldap_bind

ldap_memfree

ldap_memfree
ldap_control_free
ldap_controls_free

Purpose

Free storage allocated by the LDAP library.

Format

```
#include <ldap.h>

void ldap_memfree(
        char *mem)


void ldap_control_free(
        LDAPControl *ctrl)


void ldap_controls_free(
        LDAPControl **ctrls)
```

Parameters

Input

mem
Specifies the pointer to a character string that was previously allocated by the LDAP client library and is no longer needed by the application.

ctrl
Specifies the address of an LDAPControl structure.

ctrls
Specifies the address of an LDAPControl list, represented as a NULL-terminated array of pointers to LDAPControl structures.

Usage

In many of the LDAP programming interface calls, memory is allocated by the programming interface and returned to the application. It is the responsibility of the application to deallocate this storage when the storage is no longer needed by the application. Due to the possibility of the LDAP programming interface and the application using different heaps for dynamic storage allocation, the **ldap_memfree** API is provided for programs to use to deallocate storage that was allocated by the LDAP programming interface. It should be used to deallocate all character strings that were allocated by the programming interface and returned to the application.

For those LDAP APIs that allocate an LDAPControl structure, the **ldap_control_free** API can be used.

For those LDAP APIs that allocate an array of LDAPControl structures, the **ldap_controls_free** API can be used.

ldap_message

ldap_first_message
ldap_next_message
ldap_count_messages

Purpose

Step through the list of messages of a result chain, as returned by **ldap_result**.

Format

```
#include <ldap.h>

LDAPMessage *ldap_first_message(
        LDAP *ld,
        LDAPMessage *result)


LDAPMessage *ldap_next_message(
        LDAP *ld,
        LDAPMessage *msg)


int ldap_count_messages(
        LDAP *ld,
        LDAPMessage *result)
```

Parameters

Input

ld Specifies the LDAP handle returned by a previous call to **ldap_open**, **ldap_ssl_init**, or **ldap_init**.

result
Specifies the result returned by a call to **ldap_result** or one of the synchronous search routines (see "ldap_search" on page 73).

msg
Specifies the message returned by a previous call to **ldap_first_message** or **ldap_next_message**.

Usage

These routines are used to step through the list of messages in a result chain, as returned by **ldap_result**. For search operations, the result chain may actually include:

- Continuation reference messages
- Entry messages
- A single result message

The **ldap_count_messages** API is used to count the number of messages returned. The **ldap_msgtype** API can be used to distinguish between the different message types. Unlike **ldap_first_entry**, **ldap_first_message** will return either of the three types of messages. The other routines will return the specific type (referral or entry), skipping the others.

The **ldap_first_message** and **ldap_next_message** APIs will return NULL when no more messages exist in the result set to be returned. NULL is also returned if an error occurs while stepping through the entries. When such an error occurs, **ldap_errno** can be used to obtain the error code.

In addition to returning the number of messages contained in a chain of results, the **ldap_count_messages** API can be used to count the number of messages that remain in a chain if called with a message, entry, or reference returned by **ldap_first_message**, **ldap_next_message**, **ldap_first_entry**, **ldap_next_entry**, **ldap_first_reference** and **ldap_next_reference**.

Error Conditions

If an error occurs in **ldap_first_message** or **ldap_next_message**, the **ldap_get_errno** API can be used to obtain the error code.

If an error occurs in **ldap_count_messages**, -1 is returned, and **ldap_get_errno** can be used to obtain the error code. See “ldap_error” on page 36 for a description of possible error codes.

Related Topics

ldap_result

ldap_modify

ldap_modify
ldap_modify_ext
ldap_modify_s
ldap_modify_ext_s
ldap_mods_free

Purpose

Perform various LDAP modify operations.

Format

```
#include <ldap.h>


typedef struct ldapmod {
       int mod_op;
       char *mod_type;
       union {
            char **modv_strvals;
            struct berval **modv_bvals;
       } mod_vals;
       struct ldapmod *mod_next;
} LDAPMod;
#define mod_values mod_vals.modv_strvals
#define mod_bvalues mod_vals.modv_bvals


int ldap_modify(
        LDAP *ld,
        char *dn,
        LDAPMod *mods[])


int ldap_modify_ext(
        LDAP *ld,
        char *dn,
        LDAPMod *mods[],
        LDAPControl **serverctrls,
        LDAPControl **clientctrls,
        int *msgidp)


int ldap_modify_s(
        LDAP *ld,
        char *dn,
        LDAPMod *mods[])


int ldap_modify_ext_s(
        LDAP *ld,
        char *dn,
        LDAPMod *mods[],
        LDAPControl **serverctrls,
        LDAPControl **clientctrls)


void ldap_mods_free(
        LDAPMod **mods,
        int freemods)
```

Parameters

Input

ldap_modify

ld Specifies the LDAP handle returned by a previous call to **ldap_open**, **ldap_ssl_init**, or **ldap_init**.

dn Specifies the distinguished name (**DN**) of the entry to be modified.

mods
A NULL-terminated array of modifications to make to the entry. Each element of the *mods* array is a pointer to an LDAPMod structure.

The *mod_op* field is used to specify the type of modification to perform and should be one of **LDAP_MOD_ADD**, **LDAP_MOD_DELETE**, or **LDAP_MOD_REPLACE**. The *mod_type* and *mod_values* fields specify the attribute type to modify and a NULL-terminated array of values to add, delete, or replace respectively. The *mod_next* field is used only by the LDAP library and should be ignored by the client.

If you need to specify a non-NULL-terminated character string value (for example, to add a photo or audio attribute value), you should set *mod_op* to the logical **OR** of the operation as above (for example, **LDAP_MOD_REPLACE**) and the constant **LDAP_MOD_BVALUES**. In this case, *mod_bvalues* should be used instead of *mod_values*, and it should point to a NULL-terminated array of berval structures, as defined in the **lber.h** header file and described in “ldap_get_values” on page 46.

For **LDAP_MOD_ADD** modifications, the given values are added to the entry, creating the attribute if necessary. For **LDAP_MOD_DELETE** modifications, the given values are deleted from the entry, removing the attribute if no values remain. If the entire attribute is to be deleted, the *mod_values* field should be set to **NULL**. For **LDAP_MOD_REPLACE** modifications, the attribute will have the listed values after the modification, having been created if necessary, and deleting any existing values not in the supplied set. All modifications are performed in the order in which they are listed.

freemods
Specifies whether to deallocate the *mods* pointer. If *freemods* is nonzero, the *mods* pointer itself is deallocated as well.

serverctrls
Specifies a list of LDAP server controls. This parameter may be set to NULL. See “LDAP Controls” on page 21 for more information about server controls.

clientctrls
Specifies a list of LDAP client controls. This parameter may be set to NULL. See “LDAP Controls” on page 21 for more information about client controls.

Output

msgidp
This result parameter is set to the message ID of the request if the **ldap_modify_ext** API succeeds.

Usage

The various modify APIs are used to perform an LDAP modify operation.

The **ldap_modify_ext** API initiates an asynchronous modify operation and returns the constant **LDAP_SUCCESS** if the request was successfully sent, or another LDAP error code if not. If successful, **ldap_modify_ext** places the message ID of the request in *msgidp*. A subsequent call to **ldap_result** can be used to obtain the result of the operation. The **ldap_parse_result** API is used to extract information from the result, including any error information.

The **ldap_modify** API initiates an asynchronous modify operation and returns the message ID of the request it initiated. The result of this operation can be obtained by calling **ldap_result**, and result information can be extracted by calling **ldap_parse_result**.

For **ldap_modify** and **ldap_modify_s**, when data is supplied in a NULL-terminated character string, it is assumed to be data in the codeset of the current locale. This data will be converted to UTF-8 prior to

being passed to the LDAP server. No conversions are performed on values supplied in pointer/length format (that is, those values specified in berval structures and when **LDAP_MOD_BVALUES** is specified). All four of the LDAP modify APIs support session controls set by the **ldap_set_option** API. The **ldap_modify_ext** and **ldap_modify_ext_s** APIs both allow LDAP Version 3 server controls and client controls to be specified with the request which overrides the session controls.

Depending on how the NULL-terminated array of LDAPMod structures was allocated by the application, the **ldap_mods_free** API may or may not be useful. This API is offered as a convenience function for cleaning up previously allocated storage. When invoked, each pointer in the NULL-terminated array is deallocated and then, if *freemods* is nonzero, the *mods* pointer is also deallocated.

Error Conditions

The **ldap_modify_s** and **ldap_modify_ext_s** APIs return **LDAP_SUCCESS** if successful, otherwise an error code is returned. See “ldap_error” on page 36 for possible values.

The **ldap_modify** and **ldap_modify_ext** APIs return -1 in case of an error initiating the request. Use **ldap_get_errno** to retrieve the error value. See “ldap_error” on page 36 for possible values.

If the LDAP server is running with an SDBM database, the **ldap_modify** APIs can return **LDAP_OTHER** and have completed a partial update to an entry in RACF. The results will match what would occur if the update were done using the RACF **altuser** command. If several RACF attributes are being updated and one of them is in error, RACF reports on the error, but still updates the other attributes. The RACF message text is also returned in the result.

Related Topics

ldap_add
ldap_error

ldap_parse_result

ldap_parse_result
ldap_parse_sasl_bind_result

Purpose

LDAP APIs for extracting information from results returned by other LDAP API routines.

Format

```
#include <ldap.h>

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


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

Parameters

Input

ld Specifies the LDAP handle returned by a previous call to **ldap_open**, **ldap_ssl_init**, or **ldap_init**.

res
Specifies the result of an LDAP operation as returned by **ldap_result** or one of the synchronous LDAP API operation calls.

errcodep
Specifies a pointer to the result parameter that will be filled in with the LDAP error code field from the **LDAPMessage** message. The **LDAPResult** message is produced by the LDAP server, and indicates the outcome of the operation. NULL can be specified for *errcodep* if the **LDAPResult** message is to be ignored.

matcheddnp
Specifies a pointer to a result parameter. When **LDAP_NO_SUCH_OBJECT** is returned as the LDAP error code, this result parameter will be filled in with a distinguished name (DN) indicating how much of the name in the request was recognized by the server. **NULL** can be specified for *matcheddnp* if the matched DN is to be ignored. The matched DN string should be freed by calling **ldap_memfree**.

errmsgp
Specifies a pointer to a result parameter that is filled in with the contents of the referrals field from the **LDAPMessage** message. The error message string should be freed by calling **ldap_memfree**.

referralsp
Specifies a pointer to a result parameter that is filled in with the contents of the referrals field from the **LDAPMessage** message, indicating zero or more alternate LDAP servers where the request should be retried. The referrals array should be freed by calling **ldap_value_free**. **NULL** may be supplied for this parameter to ignore the referrals field.

serverctrlsp
Specifies a pointer to a result parameter that is filled in with an allocated array of controls copied out of the **LDAPMessage** message. The control array should be freed by calling **ldap_controls_free**.

freeit
Specifies a boolean value that determines if the LDAP result (as specified by *res*) is to be freed. Any nonzero value will result in *res* being freed after the requested information is extracted. Alternatively, the **ldap_msgfree** API can be used to free the result at a later time.

servercredp
Specifies a pointer to a result parameter. For SASL bind results, this result parameter will be filled in with the credentials returned by the server for mutual authentication (if returned). The credentials, if returned, are returned in a berval structure. **NULL** may be supplied to ignore this field.

Usage

The **ldap_parse_result** API is used to:
- Obtain the LDAP error code field associated with an **LDAPMessage** message.
- Obtain the portion of the DN that the server recognizes for a failed operation.
- Obtain the text error message associated with the error code returned in an **LDAPMessage** message.
- Obtain the list of alternate servers from the referrals field.
- Obtain the array of controls that may be returned by the server.

The **ldap_parse_sasl_bind_result** API is used to obtain server credentials, as a result of an attempt to perform mutual authentication. Both **ldap_parse_result** and **ldap_parse_sasl_bind_result** APIs ignore messages of type **LDAP_RES_SEARCH_ENTRY** and **LDAP_RES_SEARCH_REFERENCE** when looking for a result message to parse. They both return **LDAP_SUCCESS** if the result was successfully located and parsed, and an LDAP error code if not successfully parsed.

Error Conditions

The parse APIs return an LDAP error code if they encounter an error parsing the result. See “ldap_error” on page 36 for possible values.

Related Topics

ldap_error
ldap_result

ldap_rename

ldap_rename
ldap_rename_s
ldap_modrdn (deprecated)
ldap_modrdn_s (deprecated)

Purpose

Perform an LDAP rename operation.

Format

```
#include <ldap.h>

int ldap_rename(
        LDAP *ld,
        char *dn,
        char *newrdn,
        char *newparent,
        int deleteoldrdn,
        LDAPControl **serverctrls,
        LDAPControl **clientctrls,
        int *msgidp)


int ldap_rename_s(
        LDAP *ld,
        char *dn,
        char *newrdn,
        char *newparent,
        int deleteoldrdn,
        LDAPControl **serverctrls,
        LDAPControl **clientctrls)


int ldap_modrdn(
        LDAP *ld,
        char *dn,
        char *newrdn,
        int deleteoldrdn)


int ldap_modrdn_s(
        LDAP *ld,
        char *dn,
        char *newrdn,
        int deleteoldrdn)
```

Parameters

Input

ld Specifies the LDAP handle returned by a previous call to **ldap_open**, **ldap_ssl_init**, or **ldap_init**.

dn Specifies the distinguished name (DN) of the entry whose DN is to be changed. When specified with the deprecated **ldap_modrdn** and **ldap_modrdn_s** APIs, *dn* specifies the distinguished name (DN) of the entry whose relative distinguished name (RDN) is to be changed.

newrdn
Specifies the new RDN to give the entry.

newparent
Specifies the new parent, or superior entry. If this parameter is NULL, only the RDN of the entry is

changed. The root DN may be specified by passing a zero-length string, "". The *newparent* parameter should always be NULL when using Version 2 of the LDAP protocol; otherwise the server's behavior is undefined.

deleteoldrdn
If nonzero, this indicates that the old RDN value should be deleted from the entry. If zero, the attribute value is retained in the entry. With respect to the **ldap_rename** and **ldap_rename_s** APIs, this parameter only has meaning if *newrdn* is different from the old RDN.

serverctrls
Specifies a list of LDAP server controls. This parameter may be set to NULL. See "LDAP Controls" on page 21 for more information about server controls.

clientctrls
Specifies a list of LDAP client controls. This parameter may be set to NULL. See "LDAP Controls" on page 21 for more information about client controls.

Output

msgidp
This result parameter is set to the message ID of the request if the **ldap_rename** API succeeds.

Usage

In LDAP Version 2, the **ldap_modrdn** and **ldap_modrdn_s** APIs were used to change the name of an LDAP entry. They could only be used to change the least significant component of a name (the RDN or relative distinguished name). LDAP Version 3 provides the Modify DN protocol operation that allows more general name change access. The **ldap_rename** and **ldap_rename_s** APIs are used to change the name of an entry or to move a subtree of entries to a new location in the directory, and the use of the **ldap_modrdn** and **ldap_modrdn_s** APIs is deprecated.

The **ldap_rename** API initiates an asynchronous modify DN operation and returns the constant **LDAP_SUCCESS** if the request was successfully sent, or another LDAP error code if not. If successful, **ldap_rename** places the message ID of the request in *msgidp*. A subsequent call to **ldap_result** can be used to obtain the result of the operation. The **ldap_parse_result** API is used to extract information from the result, including any error information.

The synchronous **ldap_rename_s** API returns the result of the operation, either the constant **LDAP_SUCCESS** if the operation was successful, or another LDAP error code if it was not.

The LDAP rename APIs support session controls set by the **ldap_set_option** API.

The **ldap_modrdn** and **ldap_modrdn_s** APIs perform an LDAP modify RDN operation. They both change the lowest level RDN of an entry. When the RDN of the entry is changed, the value of the old RDN can be retained as an attribute type and value in the entry if desired. This is for keeping the entry inside the set of entries that match search filters which reference the attribute type of the RDN. The **ldap_modrdn** API returns the message ID of the request it initiated. The result of this operation can be obtained by calling **ldap_result**.

Error Conditions

The **ldap_rename** and **ldap_modrdn** APIs return -1 in case of an error initiating the request. Use **ldap_get_errno** to retrieve the error value. See "ldap_error" on page 36 for possible values.

The **ldap_rename_s** and **ldap_modrdn_s** APIs return **LDAP_SUCCESS** if successful, otherwise an error code is returned. See "ldap_error" on page 36 for possible values.

ldap_rename

Related Topics

ldap_error
ldap_result

ldap_result

ldap_result
ldap_msgfree
ldap_msgtype
ldap_msgid

Purpose

Wait for the result of an asynchronous LDAP operation, free the results of an operation (synchronous and asynchronous), obtain LDAP message types, and obtain the message ID of an LDAP message.

Format

```
#include <sys/time.h> /* for struct timeval definition */
#include <ldap.h>

int ldap_result(
        LDAP *ld,
        int msgid,
        int all,
        struct timeval *timeout,
        LDAPMessage **result)


int ldap_msgfree(
        LDAPMessage *msg)

int ldap_msgtype(
        LDAPMessage *msg)


int ldap_msgid(
        LDAPMessage *msg)
```

Parameters

Input

ld Specifies the LDAP handle returned by a previous call to **ldap_open**, **ldap_ssl_init**, or **ldap_init**.

msgid
Contains an invocation identifier returned when an operation was initiated. Provide the *msgid* if the result of a specific operation is required, otherwise supply **LDAP_RES_ANY**.

all For search responses, selects whether a single entry of the search should be returned or all results of the search should be returned.

A search response is made up of zero or more search entries followed by a search result. If *all* is set to 0, search entries will be returned one at a time as they come in, through separate calls to **ldap_result**. If *all* is nonzero, the search response will only be returned in its entirety, that is, after all entries and the final search result have been received. Specify **LDAP_MSG_RECEIVED** to indicate that all results retrieved so far should be returned in the result chain.

timeout
Specifies blocking for **ldap_result**. If *timeout* is not **NULL**, it specifies a maximum interval to wait for the selection to complete. If *timeout* is **NULL**, the select blocks indefinitely until the result for the operation identified by the *msgid* is received. To poll, the *timeout* parameter should be non-null, pointing to a zero-valued *timeval* structure.

msg
Pointer to a result or entry returned from **ldap_result** or from one of the synchronous LDAP search routines (see “ldap_search” on page 73).

Output

result

Contains the result of the asynchronous operation identified by *msgid*. This result should be passed to the LDAP parsing routines. See “ldap_first_entry/reference” on page 42.

The type of the result is returned in the return code. The possible result types returned are:

- **LDAP_RES_BIND**
- **LDAP_RES_SEARCH_ENTRY**
- **LDAP_RES_SEARCH_RESULT**
- **LDAP_RES_MODIFY**
- **LDAP_RES_ADD**
- **LDAP_RES_DELETE**
- **LDAP_RES_MODRDN**
- **LDAP_RES_COMPARE**
- **LDAP_RES_SEARCH_REFERENCE**
- **LDAP_RES_EXTENDED**
- **LDAP_RES_ANY**

Usage

The **ldap_result** API is used to wait for and return the result of an operation previously initiated by one of the LDAP asynchronous operation routines (for example, **ldap_search** and **ldap_modify**). Those routines return an invocation identifier upon successful initiation of the operation or -1 in case of an error. The invocation identifier is picked by the library and is guaranteed to be unique between calls to **ldap_bind** and **ldap_unbind**, or **ldap_unbind_s**. This identifier can be used to request the result of a specific operation from **ldap_result** using the *msgid* parameter.

The **ldap_result** API allocates memory for results that it receives. The memory can be deallocated by calling **ldap_msgfree**.

The **ldap_msgfree** API is used to deallocate the memory allocated for a result by **ldap_result** or the synchronous LDAP search operation routines (for example, **ldap_search_s** and **_search_s**). It takes a pointer to the result to be deallocated and returns the type of the message it deallocated.

The **ldap_msgtype** API returns the type of LDAP message, based on the LDAP message passed as input (through the *msg* parameter).

The **ldap_msgid** API returns the message ID associated with the LDAP message passed as input (through the *msg* parameter).

Error Conditions

The **ldap_result** API returns -1 if an error occurs. Use **ldap_get_errno** to retrieve the error value. Zero is returned if the *timeout* specified was exceeded. In either of these cases, the result value is meaningless.

Related Topics

ldap_search

ldap_search

ldap_search
ldap_search_s
ldap_search_ext
ldap_search_ext_s
ldap_search_st

Purpose

Perform various LDAP search operations.

Format

```
#include <sys/time.h> /* for struct timeval definition */
#include <ldap.h>
int ldap_search(
        LDAP *ld,
        char *base,
        int scope,
        char *filter,
        char *attrs[],
        int attrsonly)

int ldap_search_s(
        LDAP *ld,
        char *base,
        int scope,
        char *filter,
        char *attrs[],
        int attrsonly,
        LDAPMessage **res)

int ldap_search_ext(
        LDAP *ld,
        char *base,
        int scope,
        char *filter,
        char *attrs[],
        int attrsonly,
        LDAPControl **serverctrls,
        LDAPControl **clientctrls,
        struct timeval *timeout,
        int sizelimit,
        int *msgidp)

int ldap_search_ext_s(
        LDAP *ld,
        char *base,
        int scope,
        char *filter,
        char *attrs[],
        int attrsonly,
        LDAPControl **serverctrls,
        LDAPControl **clientctrls,
        struct timeval *timeout,
        int sizelimit,
        LDAPMessage **res)

int ldap_search_st(
        LDAP *ld,
        char *base,
        int scope,
        char *filter,
```

ldap_search

```
char *attrs[],
int attrsonly,
struct timeval *timeout,
LDAPMessage **res)
```

Parameters

Input

ld Specifies the LDAP handle returned by a previous call to **ldap_open**, **ldap_ssl_init**, or **ldap_init**.

base
Specifies the distinguished name of the entry at which to start the search. It should be in the text format described by IETF RFC 1779 *A String Representation of Distinguished Names*. Following is an example:

```
cn=Jane Doe, o=Your Company, c=US
```

scope
Specifies the scope of the search and must be one of the following:

- **LDAP_SCOPE_BASE** to search the entry named by base itself
- **LDAP_SCOPE_ONELEVEL** to search the entry's immediate children
- **LDAP_SCOPE_SUBTREE** to search the entry and all its descendents

filter
A string representation of the filter to apply in the search. Simple filters can be specified as *attributetype=attributevalue*. More complex filters are specified using a prefix notation according to the following BNF:

```
<filter> ::= '(' <filtercomp> ')'
<filtercomp> ::= <and> ; <or> ; <not> ; <simple>
<and> ::= '&' <filterlist>
<or> ::= ';' <filterlist>
<not> ::= '!' <filter>
<filterlist> ::= <filter> ; <filter> <filterlist>
<simple> ::= <attributetype> <filtertype> <attributevalue>
<filtertype> ::= '=' ; '~=' ; '<=' ; '>='
```

The '~=' construct is used to specify approximate matching. The representation for <*attributetype*> and <*attributevalue*> are as described in IETF RFC 1778, *The String Representation of Standard Attribute Syntaxes*. In addition, <*attributevalue*> can be a single asterisk (*) to achieve an attribute existence test, or can contain text and asterisks (*) interspersed to achieve substring matching.

For example, the filter

```
mail=*
```

finds any entries that have a mail attribute. The filter

```
mail=*@student.of.life.edu
```

finds any entries that have a mail attribute ending in the specified string. To put parentheses in a filter, escape them with a backslash (\) character. See IETF RFC 1558 *A String Representation of LDAP Search Filters* for a more complete description of allowable filters.

A more complicated example is:

```
(&(cn=Jane*)(sn=Doe))
```

attrs
Specifies a NULL-terminated array of character string attribute types to return from entries that match *filter*. If **NULL** is specified, all attributes are returned.

attrsonly
Specifies attribute information. If nonzero, only attribute types are returned. If zero, both attribute types and attribute values are returned.

timeout
Specifies blocking for **ldap_search_st**. If timeout is not **NULL**, it specifies a maximum interval to wait for the selection to complete. If timeout is **NULL**, the select blocks indefinitely until the result for the operation identified by the *msgid* is received. To poll, the *timeout* parameter should be non-null, pointing to a zero-valued timeval structure.

For the **ldap_search_ext** and **ldap_search_ext_s** APIs, this function specifies both the local search timeout value and the operation time limit that is sent to the server within the search request.

serverctrls
Specifies a list of LDAP server controls. This parameter may be set to NULL. See “LDAP Controls” on page 21 for more information about server controls.

clientctrls
Specifies a list of LDAP client controls. This parameter may be set to NULL. See “LDAP Controls” on page 21 for more information about client controls.

sizelimit
Specifies the maximum number of entries to return from the search. Note that the server may set a lower limit which is enforced at the server.

Output

res
Specifies the result of an LDAP operation as returned by **ldap_result** or one of the synchronous LDAP API operation calls.

msgidp
This result parameter is set to the message ID of the request if the **ldap_modify_ext** API succeeds.

Usage

The **ldap_search_ext** API initiates an asynchronous search operation and returns the constant **LDAP_SUCCESS** if the request was successfully sent, or another LDAP error code if not. If successful, **ldap_search_ext** places the message ID of the request in **msgidp*. A subsequent call to **ldap_result** can be used to obtain the results from the search. The **ldap_parse_result** API is used to extract information from the result, including any error information. In addition, use **ldap_first_entry**, **ldap_next_entry**, **ldap_first_attribute**, **ldap_next_attribute**, **ldap_get_values**, and **ldap_get_values_len** to examine results from a search.

Similar to **ldap_search_ext**, the **ldap_search** API initiates an asynchronous search operation and returns the message ID of the operation it initiated. The result of this operation can be obtained by calling **ldap_result**, and result information can be extracted by calling **ldap_parse_result**.

The **ldap_search_s** API does a synchronous search (that is, not returning until the operation completes).

The **ldap_search_st** API does a synchronous search allowing the specification of a maximum time to wait for results. The API returns when results are complete or after the timeout has passed, whichever is sooner.

All five of the LDAP search APIs support session controls set by the **ldap_set_option** API. The **ldap_search_ext** and **ldap_search_ext_s** APIs both allow LDAP Version 3 server controls and client controls to be specified with the request which overrides the session controls.

ldap_search

For **ldap_search**, **ldap_search_s**, and **ldap_search_st**, note that both read and list functionality are subsumed by these APIs. Use a filter like `objectclass=*` and a scope of **LDAP_SCOPE_BASE** to emulate read or **LDAP_SCOPE_ONELEVEL** to emulate list.

The **ldap_search_ext_s**, **ldap_search_s**, and **ldap_search_st** APIs allocate storage returned by the *res* parameter. Use **ldap_msgfree** to deallocate this storage.

There are three options in the session handle *ld* which potentially affect how the search is performed. They are:

LDAP_OPT_SIZELIMIT
: A limit on the number of entries to return from the search. A value of zero means no limit. Note that the value from the session handle is ignored when using the **ldap_search_ext** or **ldap_search_ext_s** functions.

LDAP_OPT_TIMELIMIT
: A limit on the number of seconds to spend on the search. A value of zero means no limit. Note that the value from the session handle is ignored when using the **ldap_search_ext** or **ldap_search_ext_s** APIs.

LDAP_OPT_DEREF
: One of **LDAP_DEREF_NEVER** (0x00), **LDAP_DEREF_SEARCHING**, (0x01), **LDAP_DEREF_FINDING** (0x02), or **LDAP_DEREF_ALWAYS** (0x03), specifying how aliases should be handled during the search. The **LDAP_DEREF_SEARCHING** value means aliases should be dereferenced during the search but not when locating the base object of the search. The **LDAP_DEREF_FINDING** value means aliases should be dereferenced when locating the base object but not during the search.

These options are set and queried using the **ldap_set_option** and **ldap_get_option** APIs, respectively.

Reading an Entry

LDAP does not support a read operation directly. Instead, this operation is emulated by a search with *base* set to the DN of the entry to read, scope set to **LDAP_SCOPE_BASE**, and *filter* set to `(objectclass=*)`. The *attrs* parameter optionally contains the list of attributes to return.

Listing the Children of an Entry

LDAP does not support a list operation directly. Instead, this operation is emulated by a search with *base* set to the DN of the entry to list, *scope* set to **LDAP_SCOPE_ONELEVEL**, and *filter* set to `(objectclass=*)`. The *attrs* parameter optionally contains the list of attributes to return for each child entry. If only the distinguished names of child entries are desired, the *attrs* parameter should specify a NULL-terminated array of one character string which has the value `dn`.

Error Conditions

The **ldap_search** and **ldap_search_ext** APIs return -1 in case of an error initiating the request. Use **ldap_get_errno** to retrieve the error value. See “ldap_error” on page 36 for possible values.

The **ldap_search_s**, **ldap_search_ext_s**, and **ldap_search_st** APIs return **LDAP_SUCCESS** if successful, otherwise an error code is returned. See “ldap_error” on page 36 for possible values.

Related Topics

ldap_result
ldap_error

ldap_ssl

ldap_ssl_client_init
ldap_ssl_init
ldap_ssl_start (deprecated)

Purpose

Routines for initializing the Secure Socket Layer (SSL) function for an LDAP application, and creating a secure (SSL) connection to an LDAP server.

Format

```
#include <ldap.h>

#include <ldapssl.h>

int ldap_ssl_client_init(
        char *keyring,
        char *keyring_pw,
        int ssl_timeout,
        int *pSSLReasonCode)


LDAP *ldap_ssl_init(
        char *host,
        int port,
        char *name)


int ldap_ssl_start(
        LDAP *ld,
        char *keyring,
        char *keyring_pw,
        char *name)
```

Parameters

Input

ld Specifies the LDAP pointer returned by a previous call to **ldap_ssl_init**, **ldap_init** or **ldap_open**.

host
Specifies the name of the host on which the LDAP server is running. The host parameter may contain a blank-separated list of hosts to try to connect to, and each host may optionally be of the form *host*:*port*. If present, the :*port* overrides the **ldap_ssl_init** *port* parameter. If the *host* parameter is NULL, the LDAP server will be assumed to be running on the local host.

port
Specifies the port number to which to connect. If the default IANA-assigned SSL port of 636 is desired, **LDAPS_PORT** should be specified.

keyring
Specifies the name of a key database file. The key database file typically contains one or more certificates of certificate authorities (CAs) that are trusted by the client. These types of X.509 certificates are also known as *trusted roots*. A key database file can also be used to store the client's private key or keys and associated client certificate or certificates. A private key and associated client certificate are required only if the LDAP server is configured to require client and server authentication. If the LDAP server is configured to provide only server authentication, a private key and client certificate are not required.

Note: Although still supported, use of the **ldap_ssl_start** API is discouraged (its use has been deprecated). Any application using the **ldap_ssl_start** API should only use a single key database file (per application process).

A fully-qualified path and file name is recommended. If a file name without a fully-qualified path is specified, the LDAP library will look in the current directory for the file. The key database specified here must have been created using the GSKKYMAN utility.

For more information on using GSKKYMAN to manage the contents of a key database file, see *z/OS: SecureWay Security Server LDAP Server Administration and Use* and *z/OS: System Secure Sockets Layer Programming*.

keyring_pw
Specifies the password that is used to protect the contents of the key database file. This password is important since it protects the private key stored in the key database file. The password was specified when the key database file was initially created. A NULL pointer to the password is accepted.

name
Specifies the name, or label, associated with the client private key/certificate pair in the key database file. It is used to uniquely identify a private key/certificate pair, as stored in the key database file, and may be something like: `Digital ID for Fred Smith`

If the LDAP server is configured to perform only Server Authentication, a client certificate is not required (and name can be set to NULL). If the LDAP server is configured to perform Client and Server Authentication, a client certificate is required. The *name* can be set to NULL if a default certificate/private key pair has been designated as the default (using GSKKYMAN). Similarly, *name* can be set to NULL if there is a single certificate/private key pair in the designated key database file.

ssl_timeout
Specifies the SSL timeout value in seconds. The timeout value controls the frequency with which the SSL protocol stack regenerates session keys. If *ssl_timeout* is set to 0, the default value **SSLV3_CLIENT_TIMEOUT** will be used. Otherwise, the value supplied will be used, provided it is less than or equal to 86,400 (number of seconds in a day). If *ssl_timeout* is greater than 86,400, **LDAP_PARAM_ERROR** is returned.

pSSLReasonCode
Specifies a pointer to the SSL Reason Code, which provides additional information in the event that an error occurs during initialization of the SSL stack (when **ldap_ssl_client_init** is invoked). See “ldapssl.h” on page 116 for reason codes that can be returned.

Usage

The **ldap_ssl_client_init** API is used to initialize the SSL protocol stack for an application process. It should be invoked once, prior to making any other LDAP calls. Once **ldap_ssl_client_init** has been successfully invoked, any subsequent invocations will return a return code of **LDAP_SSL_ALREADY_INITIALIZED**.

The **ldap_ssl_init** API is the SSL equivalent of **ldap_init**. It is used to initialize a secure SSL session with a server. Note that the server is not actually contacted until an operation is performed that requires it, allowing various options to be set after initialization. Once the secure connection is established for the *ld*, all subsequent LDAP messages that flow over the secure connection are encrypted, including the **ldap_simple_bind** parameters, until **ldap_unbind** is invoked.

The **ldap_ssl_init** API returns a *session handle*, a pointer to an opaque data structure that should be passed to subsequent calls that pertain to the session. These subsequent calls will return NULL if the session cannot actually be established with the server. Use **ldap_get_option** to determine why the call failed.

The LDAP session handle returned by **ldap_ssl_init** (and **ldap_init**) is a pointer to an opaque data type representing an LDAP session. The **ldap_get_option** and **ldap_set_option** APIs are used to access and set a variety of session-wide parameters. See “ldap_init” on page 49 for more information about **ldap_get_option** and **ldap_set_option**.

Note that when connecting to an LDAP Version 2 server, one of the **ldap_simple_bind** or **ldap_bind** calls must be completed before other operations can be performed on the session (with the exception of **ldap_get_option** or **ldap_set_option**). The LDAP Version 3 protocol does not require a bind operation before performing other operations.

Although still supported, the use of the **ldap_ssl_start** API is now deprecated. The **ldap_ssl_client_init** and **ldap_ssl_init** APIs should be used instead. The **ldap_ssl_start** API starts a secure connection (using SSL) to an LDAP server. The **ldap_ssl_start** API accepts the *ld* from an **ldap_open** and performs an SSL handshake to a server. The **ldap_ssl_start** API must be invoked after **ldap_open** and prior to **ldap_bind**. Once the secure connection is established for the *ld*, all subsequent LDAP messages that flow over the secure connection are encrypted, including the **ldap_bind** parameters, until **ldap_unbind** is invoked.

The following scenario depicts the recommended calling sequence where the entire set of LDAP transactions are *protected* by using a secure SSL connection, including the DN and password that flow on the **ldap_simple_bind**:

```
rc = ldap_ssl_client_init (keyfile, keyfile_pw, timeout);
ld = ldap_ssl_init(ldaphost, ldapport, label );
rc = ldap_set_option( ld, LDAP_OPT_SSL_CIPHER, &ciphers);
rc = ldap_simple_bind_s(ld, binddn, passwd);

... additional LDAP API calls

rc = ldap_unbind( ld );
```

Note that the sequence of calls for the deprecated APIs is **ldap_open** or **ldap_init**, **ldap_ssl_start**, followed by **ldap_bind**.

The following ciphers are attempted for the SSL handshake by default, in the order shown.

RC4_MD5_EXPORT
RC2_MD5_EXPORT
RC4_SHA_US
RC4_MD5_US
DES_SHA_US
3DES_SHA_US

See “ldap_init” on page 49 for more information on setting the ciphers to be used.

Options

Options are supported for controlling the nature of the secure connection. These options are set using the **ldap_set_option** API.

To specify the number of seconds for the SSL session-level timer, use:

```
ldap_set_option(ld,LDAP_OPT_SSL_TIMEOUT, &timeout)
```

where *timeout* specifies timeout in seconds. When timeout occurs, SSL re-establishes the session keys for the session for increased security.

ldap_ssl

To specify a specific cipher, or set of ciphers, to be used when negotiating with the server, use **ldap_set_option** to define a sequence of ciphers. For example, the following defines a sequence of three ciphers to be used when negotiating with the server. The first cipher that is found to be in common with the server's list of ciphers is used.

```
ldap_set_option(ld, LDAP_OPT_SSL_CIPHER,
                    (void *) LDAP_SSL_3DES_SHA_US LDAP_SSL_RC4_MD5_US);
```

The following ciphers are defined in **ldap.h**:

```
#define  LDAP_SSL_RC4_MD5_EX  "03"
#define  LDAP_SSL_RC2_MD5_EX  "06"
#define  LDAP_SSL_RC4_SHA_US  "05"
#define  LDAP_SSL_RC4_MD5_US  "04"
#define  LDAP_SSL_DES_SHA_US  "09"
#define  LDAP_SSL_3DES_SHA_US  "0A"
```

For more information on **ldap_set_option**, see "ldap_init" on page 49.

Notes

The **ldapssl.h** file contains return codes that are specific for **ldap_ssl_client_init**, **ldap_ssl_init** and **ldap_ssl_start**.

The SSL versions of these utilities include RSA software.

Related Topics

ldap_init

ldap_url

ldap_is_ldap_url
ldap_url_parse
ldap_free_urldesc
ldap_url_search
ldap_url_search_s
ldap_url_search_st

Purpose

LDAP Uniform Resource Locator (URL) routines.

Format

```
#include <sys/time.h>  /* for struct timeval definition */

#include <ldap.h>

int ldap_is_ldap_url(
        char *url)


int ldap_url_parse(
        char *url,
        LDAPURLDesc **ludpp)


typedef struct ldap_url_desc {
        char *lud_host;     /* LDAP host to contact */
        int lud_port;       /* port on host */
        char *lud_dn;       /* base for search */
        char **lud_attrs;   /* NULL-terminate list of attributes */
        int lud_scope;      /* a valid LDAP_SCOPE_...  value */
        char *lud_filter;   /* LDAP search filter */
        char *lud_string;   /* for internal use only */
        } LDAPURLDesc;


void ldap_free_urldesc(
        LDAPURLDesc *ludp)


int ldap_url_search(
        LDAP *ld,
        char *url,
        int attrsonly)


int ldap_url_search_s(
        LDAP *ld,
        char *url,
        int attrsonly,
        LDAPMessage **res)


int ldap_url_search_st(
        LDAP *ld,
        char *url,
        int attrsonly,
        struct timeval *timeout,
        LDAPMessage **res)
```

Parameters

Input

ld Specifies the LDAP handle returned by a previous call to **ldap_open**, **ldap_ssl_init**, or **ldap_init**.

url Specifies the LDAP URL.

ludp
Specifies the URL description.

attrsonly
Specifies attribute information. Set to 1 to request attributes types only. Set to 0 to request both attribute types and attribute values.

timeout
Specifies blocking for **ldap_search_st**. If *timeout* is not **NULL**, it specifies a maximum interval to wait for the selection to complete. If timeout is **NULL**, the select blocks indefinitely until the result for the operation identified by the *msgid* is received. To poll, the *timeout* parameter should be non-null, pointing to a zero-valued timeval structure.

ludpp
Points to the LDAP URL description, as returned by **ldap_url_parse**.

Output

ludpp
Points to the LDAP URL description, as returned by **ldap_url_parse**.

res
On successful completion of the search, *res* is set to point to a set of LDAPMessage structures. These should be parsed with **ldap_first_entry** and **ldap_next_entry**.

Usage

These routines support the use of LDAP URLs (Uniform Resource Locators). LDAP URLs look like this:

`ldap://hostport[:port]/dn[?attributes[?scope[?filter]]]`

where:

- *hostport* is a DNS-style host name and *port* is an optional port number
- *dn* is the base DN to be used for an LDAP search operation
- *attributes* is a comma separated list of attributes to be retrieved
- *scope* is one of these three strings:
  `base one sub (default=base)`
- *filter* is an LDAP search filter as used in a call to **ldap_search**

For example,

```
ldap://ldap.itd.umich.edu/c=US?o,description?one?o=umich
```

is an LDAP URL describing a one level search at the LDAP server running on host `ldap.itd.umich.edu` listening on the default LDAP port (389) using base distinguished name `c=US`, requesting only the organization and description attributes and applying the search filter `o=umich`.

URLs that are wrapped in angle brackets (<>) or preceded by `URL:` are also tolerated. An example of `URL:ldapurl` is:

```
URL:ldap://ldap.itd.umich.edu/c=US?o,description?one?o=umich
```

An example of <`URL:ldapurl`> is:

```
<URL:ldap://ldap.itd.umich.edu/c=US?o,description?one?o=umich>
```

The **ldap_is_ldap_url** API returns a nonzero value if *url* looks like an LDAP URL (as opposed to another type of URL). Use the **ldap_url_parse** API routine if a more thorough check is needed.

Use the **ldap_url_parse** API to check the URL more thoroughly than the **ldap_is_ldap_url** API. The **ldap_url_parse** API breaks down an LDAP URL passed in *url* into its component pieces. If successful, **LDAP_SUCCESS** is returned, an LDAP URL description is allocated, filled in, and *ludpp* is set to point to it.

The **ldap_free_urldesc** API deallocates storage allocated by **ldap_url_parse**.

The **ldap_url_search** API initiates an asynchronous LDAP search based on the contents of the *url* string. This routine acts just like **ldap_search** except that many search parameters are pulled out of the URL.

The **ldap_url_search**_s API initiates a synchronous LDAP search based on the contents of the *url* string. This routine acts just like **ldap_search_s** except that many search parameters are pulled out of the URL.

The **ldap_url_search_st** API initiates a synchronous LDAP search based on the contents of the *url* string and specifies a time-out. This routine acts just like **ldap_search_st** except that many search parameters are pulled out of the URL.

Notes

For search operations, if *hostport* is omitted, host and port for the current connection are used. If *hostport* is specified, and is different from the host and port combination used for the current connection, the search is directed to *hostport*, instead of using the current connection. In this case, the underlying referral mechanism is used to bind to *hostport*.

If the LDAP URL does not contain a search filter, the filter defaults to `objectClass=*`.

Error Conditions

If an error occurs for **ldap_url_parse**, one of the following values is returned:

LDAP_URL_ERR_NOTLDAP
URL doesn't begin with ldap://

LDAP_URL_ERR_NODN
URL has no DN (required)

LDAP_URL_ERR_BADSCOPE
URL scope string is invalid

LDAP_URL_ERR_MEM
Can't allocate memory space

The **ldap_url_search** API returns -1 in case of an error initiating the request. Use **ldap_get_errno** to retrieve the error value. See "ldap_error" on page 36 for possible values.

The **ldap_url_search**_s and **ldap_url_search_st** APIs return **LDAP_SUCCESS** if successful, otherwise an error code is returned. See "ldap_error" on page 36 for possible values.

Related Topics

ldap_search

Chapter 3. LDAP Operation Utilities

Several utility programs are provided that implement some of the LDAP APIs. These utilities provide a way to add, modify, search and delete entries in any server accepting LDAP protocol requests.

Each of the following programs can be run from the z/OS shell, or from TSO:
- **ldapadd**
- **ldapmodify**
- **ldapmodrdn**
- **ldapsearch**
- **ldapdelete**

Running the LDAP Operation Utilities in the z/OS Shell

In order to run any of these utilities in the shell, some environment variables need to be set properly. Ensure that **/bin** is included in the **PATH** environment variable. Also, make sure **STEPLIB** is set to *GLDHLQ*.SGLDLNK.

Each of these utilities accepts many possible parameters. See “Using the Command Line Utilities” on page 86 for a complete explanation of the parameters that can be supplied to each of the operation utility programs.

Note: When using these utilities to communicate with an z/OS LDAP Server operating in multi-server mode with dynamic workload management enabled (see the configuring chapter in *z/OS: SecureWay Security Server LDAP Server Administration and Use* for additional information about LDAP server operating modes), the *hostname* value in the preceding commands should be in the form *group_name.sysplex_domain_name*, where *group_name* is the name of the **sysplexGroupName** identified in the server configuration files and *sysplex_domain_name* is the name or alias of the sysplex domain in which the servers operate.

Running the LDAP Operation Utilities in TSO

The LDAP operation utilities can be run from TSO. In order to do this, some elements of the environment need to be set up to locate the LDAP programs.

First, the PDS (*GLDHLQ*.SGLDLNK) where the LDAP server load modules were installed needs to be specified in one of **LINKLIB**, **LPALIB** or **TSOLIB**. Second, the PDS (*GLDHLQ*.SGLDEXEC) containing the CLISTs needed to run the utilities must be available in **SYSEXEC**.

Once this setup is complete, running these utilities follows the same syntax as would be used if running them in z/OS, except that the program names are eight characters or less. To run these utilities from TSO, use the following names:

z/OS Shell Name	TSO Name
ldapadd	ldapadd
ldapmodify	ldapmdfy
ldapmodrdn	ldapmrdn
ldapsearch	ldapsrch
ldapdelete	ldapdlet

Using the Command Line Utilities

The **ldapdelete**, **ldapmodify**, **ldapadd**, **ldapmodrdn**, and **ldapsearch** utilities all use the **ldap_bind** API. When bind is invoked, several results can be returned. Following are bind results using various combinations of user IDs and passwords.

1. If specifying the administration DN, the password must be correctly specified or the bind will not be successful.
2. If a null DN is specified, or a 0 length DN is specified, you will receive unauthenticated access.
3. If a DN is specified, and it is non-null, a password must also be specified or an error will be returned.
4. If a DN and password are specified, but it does not fall under any suffix in the directory, a referral will be returned.
5. If a DN and password are specified, and are correct, the user is bound with that identity.
6. If a DN and password are specified, but the DN does not exist, unauthenticated access will be given.
7. If a DN and password are specified, and the DN exists, but the object does not have *userpassword*, an error message will be returned.
8. If a DN and password are specified, and the DN exists, but the password is of 0 length, then unauthenticated access will be given.

SSL Information for LDAP Utilities

The contents of a client's key database file is managed with the **gskkyman** utility. See *z/OS System Secure Sockets Layer Programming Guide* for information about the **gskkyman** utility. The **gskkyman** utility is used to define the set of trusted certification authorities (CAs) that are to be trusted by the client. By obtaining certificates from trusted CAs, storing them in the key database file, and marking them as trusted, you can establish a trust relationship with LDAP servers that use certificates issued by one of the CAs that are marked as trusted.

If the LDAP servers accessed by the client use server authentication, it is sufficient to define one or more trusted root certificates in the key database file. With server authentication, the client can be assured that the target LDAP server has been issued a certificate by one of the trusted CAs. In addition, all LDAP transactions that flow over the SSL connection with the server are encrypted, including the LDAP credentials that are supplied on the **ldap_bind** API.

For example, if the LDAP server is using a high-assurance VeriSign certificate, you should obtain a CA certificate from VeriSign, receive it into your key database file, and mark it as trusted. If the LDAP server is using a self-signed **gskkyman** server certificate, the administrator of the LDAP server can supply you with a copy of the server's certificate request file. Receive the certificate request file into your key database file and mark it as trusted.

Using this utility without the **-Z** parameter and calling the SSL-defined port on an LDAP server (a non-SSL call to an SSL port) is not supported. Also, an SSL call to a non-SSL port is not supported.

ldapdelete Utility

Purpose

The **ldapdelete** utility is a shell-accessible interface to the **ldap_delete** API.

The **ldapdelete** utility opens a connection to an LDAP server, binds, and deletes one or more entries. If one or more *dn* arguments are provided, entries with those DNs are deleted. If no *dn* arguments are provided, a list of DNs is read from standard input (**<***entryfile*) or from *file* if the **-f** flag is used.

Format

ldapdelete [*options*] {**-f** *file* | < **entryfile** | *dn...* }

Parameters

options

The following table shows the *options* you can use for the **ldapdelete** utility:

Table 4. ldapdelete Options

Option	Description
-?	Print this text.
-V *version*	Specify the LDAP protocol level the client should use. The value for *version* can be **2** or **3**. The default is **2**.
-c	Continuous operation mode. Errors are reported, but **ldapdelete** will continue with deletions. The default is to exit after reporting an error.
-n	Show what would be done, but do not actually delete entries. Useful for debugging in conjunction with **-v**.
-v	Use verbose mode, with many diagnostics written to standard output.
-R	Do not automatically follow referrals.
-M	Manage referral objects as normal entries. This requires a protocol level of 3 (specify the **-V 3** parameter).
-d *debuglevel*	Set the LDAP debugging level to *debuglevel*. *z/OS: SecureWay Security Server LDAP Server Administration and Use* lists the possible values for *debuglevel*.
-D *binddn*	Use *binddn* to bind to the LDAP directory. The *binddn* parameter should be a string-represented DN. The default is a NULL string.
-w *passwd*	Use *passwd* as the password for simple authentication. The default is a NULL string.
-h *ldaphost*	Specify the host on which the LDAP server is running. The default is the local host. When the target host is a z/OS LDAP server operating in multi-server mode with dynamic workload management enabled (see the configuring chapter in *z/OS: SecureWay Security Server LDAP Server Administration and Use* for additional information about LDAP server operating modes), the *ldaphost* value should be in the form *group_name.sysplex_domain_name*, where *group_name* is the name of the **sysplexGroupName** identified in the server configuration file and *sysplex_domain_name* is the name or alias of the sysplex domain in which the target server operates.
-p *ldapport*	Specify the TCP port where the LDAP server is listening. The default LDAP port is 389. If not specified and **-Z** is specified, the default LDAP SSL port 636 is used.
-Z	Use a secure SSL connection to communicate with the LDAP server. The **-Z** option is not supported by non-SSL versions of this tool.

Table 4. ldapdelete Options (continued)

Option	Description
-K *keyfile*	Specify the name of the SSL key database file. If the key database file is not in the current directory, specify the fully-qualified key database file name. If a key database file name is not specified, this utility looks for the presence of the **SSL_KEYRING** environment variable with an associated file name. Otherwise, no key database file will be used for server authentication and default trusted certification authority roots will be used. The key database file typically contains one or more certificates of certification authorities (CAs) that are trusted by the client. These types of X.509 certificates are also known as trusted roots. For more information on managing an SSL key database file, see *z/OS: System Secure Sockets Layer Programming* for a description on the use of the **gskkyman** utility to manage the contents of a key database file. Also see the SSL section in *z/OS: SecureWay Security Server LDAP Server Administration and Use* for more information about SSL and certificates. This parameter is ignored if **-Z** is not specified.
-P *keyfilepw*	Specify the key database file password. This password is required to access the encrypted information in the key database file (including the private key). If the key database file does not contain a private key, which is possible on the LDAP client, then the key database file may have been created without a password. In this case, there is no need to specify a password here. This parameter is ignored if **-Z** is not specified.
-N *keyfiledn*	Specify the certificate name in the key database file.

-f *file*
Read a series of lines from *file*, performing one LDAP delete for the DN on each line.

entryfile
Specify a file containing DNs to delete on consecutive lines.

dn Specify distinguished name (DN) of an entry to delete. You can specify one or more *dn* arguments. Each *dn* should be a string-represented DN.

Examples

The following command:

```
ldapdelete "cn=Delete Me, o=My Company, c=US"
```

attempts to delete the entry named with **commonName** `Delete Me` directly below `My Company` organizational entry. It may be necessary to supply a *binddn* and *passwd* for deletion to be allowed (see the **-D** and **-w** options).

Notes

If no *dn* arguments are provided, the **ldapdelete** command will wait to read a list of DNs from standard input. To break out of the wait, use <Ctrl-C> or <Ctrl-D> .

SSL Note

See “SSL Information for LDAP Utilities” on page 86.

Diagnostics

Exit status is 0 if no errors occur. Errors result in a nonzero exit status and a diagnostic message being written to standard error.

ldapmodify and ldapadd Utilities

The **ldapmodify** utility is a shell-accessible interface to the **ldap_modify** and **ldap_add** APIs. The **ldapadd** command is implemented as a renamed version of **ldapmodify**. When invoked as **ldapadd**, the -a (add new entry) flag is turned on automatically.

The **ldapmodify** utility opens a connection to an LDAP server, binds, and modifies or adds entries. The entry information is read from standard input or from *file* through the use of the **-f** option.

Format

ldapmodify | **ldapadd** [*options*]

Parameters

options
The following table shows the options you can use for the **ldapmodify** and **ldapadd** utilities:

Table 5. ldapmodify and ldapadd Options

Option	Description
-?	Print this text.
-V *version*	Specify the LDAP protocol level the client should use. The value for *version* can be **2** or **3**. The default is **2**.
-c	Continuous operation mode. Errors are reported, but **ldapmodify** will continue with modifications. The default is to exit after reporting an error.
-n	Show what would be done, but do not actually modify entries. Useful for debugging in conjunction with **-**v.
-v	Use verbose mode, with many diagnostics written to standard output.
-R	Do not automatically follow referrals.
-M	Manage referral objects as normal entries. This requires a protocol level of 3 (specify the **-V 3** parameter).
-a	Add new entries. The default for **ldapmodify** is to modify existing entries. If invoked as **ldapadd**, this flag is always set.
-b	Assume that any values that start with a slash (/) are binary values and that the actual value is in a file whose path is specified in the place where values normally appear.
-r	Replace existing values by default.
-F	Force application of all changes regardless of the contents of input lines that begin with **replica**: (by default, **replica**: lines are compared against the LDAP server host and port in use to decide if a replication log record should actually be applied).
-d *debuglevel*	Set the LDAP debugging level to *debuglevel*. *z/OS: SecureWay Security Server LDAP Server Administration and Use* lists the possible values for *debuglevel*.
-f *file*	Read the entry modification information from *file* instead of from standard input.
-D *binddn*	Use *binddn* to bind to the LDAP directory. The *binddn* should be a string-represented DN. The default is a NULL string.
-w *passwd*	Use *passwd* as the password for simple authentication. The default is a NULL string.

Table 5. ldapmodify and ldapadd Options (continued)

Option	Description
-h *ldaphost*	Specify the host on which the LDAP server is running. The default is the local host. When the target host is a z/OS LDAP server operating in multi-server mode with dynamic workload management enabled (see the configuring chapter in *z/OS: SecureWay Security Server LDAP Server Administration and Use* for additional information about LDAP server operating modes), the *ldaphost* value should be in the form *group_name.sysplex_domain_name*, where *group_name* is the name of the **sysplexGroupName** identified in the server configuration file and *sysplex_domain_name* is the name or alias of the sysplex domain in which the target server operates.
-p *ldapport*	Specify the TCP port where the LDAP server is listening. The default LDAP port is 389. If not specified and **-Z** is specified, the default LDAP SSL port 636 is used.
-Z	Use a secure SSL connection to communicate with the LDAP server. The **-Z** option is not supported by non-SSL versions of this tool.
-K *keyfile*	Specify the name of the SSL key database file. If the key database file is not in the current directory, specify the fully-qualified key database file name. If a key database file name is not specified, this utility will look for the presence of the **SSL_KEYRING** environment variable with an associated file name. Otherwise, no key database file will be used for server authentication and default trusted certification authority roots will be used. The key database file typically contains one or more certificates of certification authorities (CAs) that are trusted by the client. These types of X.509 certificates are also known as trusted roots. For more information on managing an SSL key database file, see *z/OS: System Secure Sockets Layer Programming* for a description on the use of the **gskkyman** utility to manage the contents of a key database file. Also see the SSL section in the *z/OS: SecureWay Security Server LDAP Server Administration and Use* for more information about SSL and certificates. This parameter is ignored if **-Z** is not specified.
-P *keyfilepw*	Specify the key database file password. This password is required to access the encrypted information in the key database file (including the private key). If the key database file does not contain a private key, which is possible on the LDAP client, then the key database file may have been created without a password. In this case, there is no need to specify a password here. This parameter is ignored if **-Z** is not specified.
-N *keyfiledn*	Specify the certificate name in the key database file.

Input Modes

The **ldapmodify** command as well as the **ldapadd** command accept two forms of input. The type of input is determined by the format of the first input line supplied to **ldapmodify** or **ldapadd**.

Note: The **ldapadd** command is equivalent to invoking the **ldapmodify -a** command.

The first line of input to the **ldapmodify** command (or **ldapadd** command) must denote the distinguished name of a directory entry to add or modify. This input line must be of the form:

```
dn:distinguished_name
```

or

```
distinguished_name
```

where **dn**: is a literal string and *distinguished_name* is the distinguished name of the directory entry to modify (or add). If **dn**: is found, the input mode is set to *LDIF mode*. If it is not found, the input mode is set to *modify mode*.

Note: The **ldapmodify** and **ldapadd** utilities do not support base64 encoded distinguished names.

LDIF Mode: When using LDIF mode style input, attribute types and values are delimited by colons (or double colons (::)). Furthermore, individual changes to attribute values are delimited with a **changetype:** input line. The general form of input lines for LDIF mode is:

```
change_record
<blank line>
change_record
<blank line>
.
.
.
```

An input file in LDIF mode consists of one or more *change_record* sets of lines which are separated by a single blank line. Each *change_record* has the following form:

dn:*distinguished_name* [**changetype**:{**modify**|**add** |**modrdn**|**delete**}] [*change_clause* . . .]

Thus, a *change_record* consists of a line indicating the distinguished name of the directory entry to be modified, an optional line indicating the type of modification to be performed against the directory entry, along with one or more *change_clause* sets of lines. If the **changetype** line is omitted, then the change type is assumed to be **modify** unless the command invocation was **ldapmodify -a** or **ldapadd**, in which case the **changetype** is assumed to be **add**.

When the change type is **modify**, each *change_clause* is defined as a set of lines of the form:

```
add:x
{attrtype}{sep}{value}
.
.
.
-
```

or

```
replace:x
{attrtype}{sep}{value}
.
.
.
-
```

or

```
delete:{attrtype}
-
```

or

```
{attrtype}{sep}{value}
.
.
.
```

Specifying **replace** replaces all existing values for the attribute with the specified set of attribute values. Specifying **add** adds to the existing set of attribute values.

If an **add:x**, **replace:x**, or **delete:x** line (a change indicator) is specified, a line containing a hyphen (-) is expected as a closing delimiter for the changes. Attribute-value pairs are expected on the input lines that

are found between the change indicator and hyphen line. If the change indicator line is omitted, the change is assumed to be **add** for the attribute values specified. However, if the **-r** option is specified on **ldapmodify**, then the *change_clause* is assumed to be **replace**. The separator, *sep*, can be either a single colon (:) or double colon (::). Any white space between the separator and the attribute value is ignored. Attribute values can be continued across multiple lines by using a single space character as the first character of the next line of input. If a double colon is used as the separator, then the input is expected to be in so-called **base64**. format. This format is an encoding that represents every three binary bytes with four text characters. Refer to the **base64encode()** function in **/usr/lpp/ldap/examples/line64.c** for an implementation of this encoding.

Multiple attribute values are specified using multiple {*attrtype*}{*sep*}{*value*} specifications.

When the change type is **add**, each *change_clause* is defined as a set of lines of the form:

{*attrtype*}{*sep*}{*value*}

As with change type of **modify**, the separator, *sep*, can be either a single colon (:) or double colon (::). Any white space between the separator and the attribute value is ignored. Attribute values can be continued across multiple lines by using a single space character as the first character of the next line of input. If a double colon is used as the separator, then the input is expected to be in so-called **base64** format.

When the change type is **modrdn**, each *change_clause* is defined as a set of lines of the form:

newrdn:*value*
deleteoldrdn:{**0**|**1**}

These are the parameters you can specify on a modify RDN LDAP operation. The value for the **newrdn** setting is the new RDN to be used when performing the modify RDN operation. Specify 0 for the value of the **deleteoldrdn** setting in order to save the attribute in the old RDN and specify 1 to remove the attribute values in the old RDN.

When the change type is **delete**, no *change_clause* is specified.

The LDIF mode of input allows for almost any form of update to the LDAP directory to be accomplished. The one operation that cannot be performed by the **ldapmodify** command is the deletion of individual attribute values.

Modify Mode: The modify mode of input to the **ldapmodify** or **ldapadd** commands is not as flexible as the LDIF mode. However, it is sometimes easier to use than the LDIF mode.

When using modify mode style input, attribute types and values are delimited by an equal sign (=). The general form of input lines for modify mode is:

```
change_record
<blank line>
change_record
<blank line>
.
.
.
```

An input file in modify mode consists of one or more *change_record* sets of lines which are separated by a single blank line. Each *change_record* has the following form:

```
distinguished_name
[+|-]{attrtype} ={value_line1[\
value_line2[\
```

```
...value_lineN]]}
.
.
.
```

Thus, a *change_record* consists of a line indicating the distinguished name of the directory entry to be modified along with one or more attribute modification lines. Each attribute modification line consists of an optional add or delete indicator, an attribute type, and an attribute value. If a plus sign (+) is specified, then the modification type is set to **add**. If a hyphen (-) is specified then the modification type is set to **delete**. If the add or delete indicator is not specified, then the modification type is set to **add** unless the **-r** option is used, in which case the modification type is set to **replace**. Any leading or trailing white-space characters are removed from attribute values. If trailing white-space characters are required for attribute values, then the LDIF mode of input must be used. Lines are continued using a backslash (\) as the last character of the line. If a line is continued, the backslash character is removed and the succeeding line is appended directly after the character preceding the backslash character. The new-line character at the end of the input line is not retained as part of the attribute value.

Multiple attribute values are specified using multiple *attrtype=value* specifications.

The modify mode is not as flexible as the LDIF mode of input. However, it does allow deletion of individual attribute values which the LDIF mode does not support.

Input Mode Examples: Here are some examples of valid input for the **ldapmodify** command.

Adding a New Entry:

```
dn:cn=Tim Doe, ou=Your Department, o=Your Company, c=US
changetype:add
cn: Tim Doe
sn: Doe
objectclass: organizationalperson
objectclass: person
objectclass: top
```

This example adds a new entry into the directory using name `cn=Tim Doe, ou=Your Department, o=Your Company, c=US`, assuming **ldapadd** or **ldapmodify -a** is invoked.

Adding Attribute Types:

```
dn:cn=Tim Doe, ou=Your Department, o=Your Company, c=US
changetype:modify
add:x
telephonenumber: 888 555 1234
registeredaddress: td@yourcompany.com
registeredaddress: ttd@yourcompany.com
-
```

This example adds two new attribute types to the existing entry. Note that the **registeredaddress** attribute is assigned two values.

Changing the Entry Name:

```
dn: cn=Tim Doe, ou=Your Department, o=Your Company, c=US
changetype:modrdn
newrdn: cn=Tim Tom Doe
deleteoldrdn: 0
```

This example changes the name of the existing entry to `cn=Tim Tom Doe, ou=Your Department, o=Your Company, c=US`. The old RDN, `cn=Tim Doe`, is retained as an additional attribute value of the **cn** attribute.

Replacing Attribute Values:

```
dn: cn=Tim Tom Doe, ou=Your Department, o=Your Company, c=US
changetype:modify
replace:x
telephonenumber: 888 555 4321
registeredaddress: TTD@YOURCOMPANY.COM
-
```

This example replaces the attribute values for the **telephonenumber** and **registeredaddress** attributes with the specified attribute values.

Deleting and Adding Attributes:

```
dn:cn=Tim Tom Doe, ou=Your Department, o=Your Company, c=US
changetype:modify
add:x
description: This is a very long attribute
  value that is continued on a second line.
   Note the spacing at the beginning of the
  continued lines in order to signify that
  the line is continued.
-
delete: phone
-
```

This example deletes the **telephonenumber** attribute and adds a **description** attribute. The description attribute value spans multiple lines.

Deleting an Entry:

```
dn:cn=Tim Tom Doe, ou=Your Department, o=Your Company, c=US
changetype:delete
```

This example deletes the directory entry with name `cn=Tim Tom Doe, ou=Your Department, o=Your Company, c=US`.

Adding a New Entry:

```
cn=Tim Doe, ou=Your Department, o=Your Company, c=US
cn=Tim Doe
sn=Doe
objectclass=organizationalperson
objectclass=person
objectclass=top
```

This example adds a new entry into the directory using name `cn=Tim Doe, ou=Your Department, o=Your Company, c=US`

Adding a New Attribute Type:

```
cn=Tim Doe, ou=Your Department, o=Your Company, c=US
+telephonenumber=888 555 1234
+registeredaddress=td@yourcompany.com
+registeredaddress=ttd@yourcompany.com
```

This example adds two new attribute types to the existing entry. Note that the **registeredaddress** attribute is assigned two values.

Replacing Attribute Values:

```
cn=Tim Doe, ou=Your Department, o=Your Company, c=US
telephonenumber=888 555 4321
registeredaddress=TTD@YOURCOMPANY.COM
```

Assuming that the command invocation was:

```
ldapmodify -r ...
```

this example replaces the attribute values for the **telephonenumber** and **registeredaddress** attributes with the specified attribute values. If the **-r** command line option was not specified, then the attribute values are added to the existing set of attribute values.

Deleting an Attribute Type:

```
cn=Tim Doe, ou=Your Department, o=Your Company, c=US
-registeredaddress=ttd@yourcompany.com
```

This example deletes a single **registeredaddress** attribute value from the existing entry.

Adding an Attribute:

```
cn=Tim Doe, ou=Your Department, o=Your Company, c=US
+description=This is a very long attribute \
value that is continued on a second line.  \
Note the backslash at the end of the line to \
be continued in order to signify that \
the line is continued.
```

This example adds a **description** attribute. The **description** attribute value spans multiple lines.

Input Format

An alternative input format is supported for compatibility with older versions of **ldapmodify**. This format consists of one or more entries separated by blank lines, where each entry looks like:

```
distinguished_name
attr=value
[attr=value ... ]
```

where *attr* is the name of the attribute and *value* is the value. By default, values are added. If the **-r** command-line flag is given, the default is to replace existing values with the new one. Note that it is permissible for a given attribute to appear more than once (for example, to add more than one value for an attribute). Also note that you can use a trailing backslash (\) to continue values across lines and preserve new lines in the value itself. The *attr* should be preceded by a dash (-) to remove a value. The equal sign (=) and value should be omitted to remove an entire attribute. The *attr* should be preceded by a plus sign (+) to add a value in the presence of the **-r** flag.

Examples

Following are some **ldapmodify** and **ldapadd** examples:

- Assuming that the file **/tmp/entrymods** exists and has the contents:

  ```
  dn: cn=Modify Me, o=My Company, c=US
  changetype: modify
  replace: mail
  mail: modme@MyCompany.com
  -
  add: title
  title: Vice President
  -
  add: jpegPhoto
  jpegPhoto: /tmp/modme.jpeg
  -
  delete: description
  -
  ```

  the command:

  ```
  ldapmodify -b -r -f /tmp/entrymods
  ```

replaces the contents of the `Modify Me` entry's **mail** attribute with the value `modme@MyCompany.com`, adds a **title** of `Vice President`, and the contents of the file **/tmp/modme.jpeg** as a **jpegPhoto**, and completely removes the **description** attribute. The same modifications as above can be performed using the older **ldapmodify** input format:

```
cn=Modify Me, o=My Company, c=US
mail=modme@MyCompany.com
+title=Vice President
+jpegPhoto=/tmp/modme.jpeg
-description
```

- Assuming that the file **/tmp/newentry** exists and has the contents:

```
dn: cn=Joe Smith, o=My Company, c=US
objectClass: person
cn: Joseph Smith
cn: Joe Smith
sn: Smith
title: Manager
mail: jsmith@jsmith.MyCompany.com
uid: jsmith
```

the command:

```
ldapadd -f /tmp/newentry
```

adds a new entry for Joe Smith, using the values from the file **/tmp/newentry**.

- Assuming that the file **/tmp/newentry** exists and has the contents:

```
dn: cn=Joe Smith, o=My Company, c=US
changetype: delete
```

the command:

```
ldapmodify -f /tmp/newentry
```

removes Joe Smith's entry.

- Assuming `hostA` contains the referral object:

```
dn: o=ABC,c=US
ref: ldap://hostB:390/o=ABC,c=US
objectclass: referral
```

and `hostB` contains the organization object:

```
dn: o=ABC,c=US
o: ABC
objectclass: organization
telephoneNumber: 123-4567
```

and the file **/tmp/refmods** contains:

```
dn: o=ABC,c=US
changetype: modify
replace: ref
ref: ldap://hostB:391/o=ABC,c=US
-
```

and the file **/tmp/ABCmods** contains:

```
dn: o=ABC,c=US
changetype: modify
add: telephoneNumber
telephoneNumber: 123-1111
-
```

the command:

```
ldapmodify -h hostA -r -V 3 -M -f /tmp/refmods
```

replaces the **ref** attribute value of the referral object `o=ABC,c=US` in `hostA`, changing the TCP port address in the URL from 390 to 391.

The command:

```
ldapmodify -h hostB -p 391 -f /tmp/ABCmods
```

adds the **telephoneNumber** attribute value `123-1111` to `o=ABC,c=US` in `hostB`.

SSL Note

See “SSL Information for LDAP Utilities” on page 86.

Diagnostics

Exit status is 0 if no errors occur. Errors result in a nonzero exit status and a diagnostic message being written to standard error.

ldapmodrdn Utility

Purpose

The **ldapmodrdn** utility is a shell-accessible interface to the **ldap_modrdn** API.

The **ldapmodrdn** utility opens a connection to an LDAP server, binds, and modifies the RDN of entries. The entry information is read from standard input (**<***entryfile*), from *file* through the use of the **-f** option, or from the command-line pair *dn* and *newrdn*. The entries must be leaf entries.

Format

```
ldapmodrdn [options] {-f file | < entryfile | dn newrdn }
```

Parameters

options

The following table shows the *options* you can use for the **ldapmodrdn** utility:

Table 6. ldapmodrdn Options

Option	Description
-?	Print this text.
-V *version*	Specify the LDAP protocol level the client should use. The value for *version* can be **2** or **3**. The default is **2**.
-c	Continuous operation mode. Errors are reported, but **ldapmodrdn** will continue with modifications. The default is to exit after reporting an error.
-n	Show what would be done, but do not actually change entries. Useful for debugging in conjunction with **-v**.
-r	Remove old RDN values from the entry. Default is to keep old values.
-v	Use verbose mode, with many diagnostics written to standard output.
-R	Do not automatically follow referrals.
-M	Manage referral objects as normal entries. This requires a protocol level of 3 (specify the **-V 3** parameter).
-d *debuglevel*	Set the LDAP debugging level to *debuglevel*. *z/OS: SecureWay Security Server LDAP Server Administration and Use* lists the possible values for *debuglevel*.
-D *binddn*	Use *binddn* to bind to the LDAP directory. The *binddn* should be a string-represented DN. The default is a NULL string.
-w *passwd*	Use *passwd* as the password for simple authentication. The default is a NULL string.
-h *ldaphost*	Specify the host on which the LDAP server is running. The default is the local host. When the target host is a z/OS LDAP server operating in multi-server mode with dynamic workload management enabled (see the configuring chapter in the *z/OS: SecureWay Security Server LDAP Server Administration and Use* for additional information about LDAP server operating modes), the *ldaphost* value should be in the form *group_name.sysplex_domain_name*, where *group_name* is the name of the **sysplexGroupName** identified in the server configuration file and *sysplex_domain_name* is the name or alias of the sysplex domain in which the target server operates.
-p *ldapport*	Specify the TCP port where the LDAP server is listening. The default LDAP port is 389. If not specified and **-Z** is specified, the default LDAP SSL port 636 is used.
-Z	Use a secure SSL connection to communicate with the LDAP server. The **-Z** option is not supported by non-SSL versions of this tool.

Table 6. ldapmodrdn Options (continued)

Option	Description
-K *keyfile*	Specify the name of the SSL key database file. If the key database file is not in the current directory, specify the fully-qualified key database file name. If a key database file name is not specified, this utility will look for the presence of the **SSL_KEYRING** environment variable with an associated file name. Otherwise, no key database file will be used for server authentication and default trusted certification authority roots will be used. The key database file typically contains one or more certificates of certification authorities (CAs) that are trusted by the client. These types of X.509 certificates are also known as trusted roots. For more information on managing an SSL key database file, see *z/OS: System Secure Sockets Layer Programming* for a description on the use of the **gskkyman** utility to manage the contents of a key database file. Also see the SSL section in *z/OS: SecureWay Security Server LDAP Server Administration and Use* for more information about SSL and certificates. This parameter is ignored if **-Z** is not specified.
-P *keyfilepw*	Specify the key database file password. This password is required to access the encrypted information in the key database file (including the private key). If the key database file does not contain a private key, which is possible on the LDAP client, then the key database file may have been created without a password. In this case, there is no need to specify a password here. This parameter is ignored if **-Z** is not specified.
-N *keyfiledn*	Specify the certificate name in the key database file.

-f *file*
Read the entry modification information from *file* instead of from standard input or the command line (by specifying *dn* and *newrdn*). Standard input can also be supplied from a file (<*entryfile*).

entryfile
Specify a file containing the old DN and new RDN on consecutive lines.

dn Specify the DN of the entry to change.

newrdn
Specify the new RDN for the entry.

Input Format

If the command-line arguments *dn* and *newrdn* are given, *newrdn* replaces the RDN of the entry specified by the DN, *dn*. Otherwise, the contents of *file* (or standard input if no **-f** flag is given) should consist of one or more entries.

```
Distinguished Name (DN)
Relative Distinguished Name (RDN)
```

One or more blank lines may be used to separate each DN/RDN pair.

Examples

Assuming that the file **/tmp/entrymods** exists and has the contents:

```
cn=Modify Me, o=My Company, c=US
cn=The New Me
```

the command:

```
ldapmodrdn -r -f /tmp/entrymods
```

ldapmodrdn

changes the RDN of the `Modify Me` entry from `Modify Me` to `The New Me` and the old CN, `Modify Me` is removed.

SSL Note

See “SSL Information for LDAP Utilities” on page 86.

Diagnostics

Exit status is 0 if no errors occur. Errors result in a nonzero exit status and a diagnostic message being written to standard error.

ldapsearch Utility

Purpose

The **ldapsearch** utility is a shell-accessible interface to the **ldap_search** routine.

The **ldapsearch** utility opens a connection to an LDAP server, binds, and performs a search using the *filter*. If **ldapsearch** finds one or more entries, the *attributes* specified are retrieved and the entries and values are printed to standard output.

Note: Use of the approximate filter (~=) is not supported on a z/OS server.

Format

```
ldapsearch [options] filter [attributes...]
```

Parameters

options

The following table shows the *options* you can use for the **ldapsearch** utility:

Table 7. ldapsearch Options

Option	Description
-?	Print this text.
-V *version*	Specify the LDAP protocol level the client should use. The value for *version* can be **2** or **3**. The default is **2**.
-S *method*	Specify the bind method to use. The default is **SIMPLE**. You can also specify **EXTERNAL** to indicate that a certificate (SASL external) bind is requested. The **EXTERNAL** method requires a protocol level of 3 (specify the **-V 3** parameter). You must also specify **-Z**, **-K**, and **-P** to use certificate bind. If there is more than one certificate in the key database file, use **-N** to specify the certificate or the default certificate will be used.
-n	Show what would be done, but do not actually perform the search. Useful for debugging in conjunction with **-v**.
-v	Run in verbose mode, with many diagnostics written to standard output.
-t	Write retrieved values to a set of temporary files. This option assumes values are nontextual (binary), such as **jpegPhoto** or **audio**. There is no character set translation performed on the values.
-A	Retrieve attributes only (no values). This is useful when you just want to see if an attribute is present in an entry and are not interested in the specific values.
-B	Do not suppress display of non-printable values. This is useful when dealing with values that appear in alternate character sets such as ISO-8859.1. This option is implied by the **-L** option.
-C	Do not suppress display of printable non-ASCII values (similar to the **-B** option). Values are displayed in the local codepage.
-L	Display search results in LDIF format. This option also turns on the **-B** option, and causes the **-F** option to be ignored.
-R	Do not automatically follow referrals.
-M	Manage referral objects as normal entries. This requires a protocol level of 3 (specify the **-V 3** parameter).
-d *debuglevel*	Set the LDAP debugging level to *debuglevel*. *z/OS: SecureWay Security Server LDAP Server Administration and Use* lists the possible values for *debuglevel*.
-F *sep*	Use *sep* as the field separator between attribute names and values. The default separator is an equal sign (=), unless the **-L** flag has been specified, in which case this option is ignored.

ldapsearch

Table 7. ldapsearch Options (continued)

Option	Description
-f *file*	Read a series of lines from *file*, performing one LDAP search for each line. In this case, the *filter* given on the command line is treated as a pattern where the first occurrence of **%s** is replaced with a line from *file*. If *file* is a single hyphen (-) character, then the lines are read from standard input.
-b *searchbase*	Use *searchbase* as the starting point for the search instead of the default. If **-b** is not specified, this utility examines the **LDAP_BASEDN** environment variable for a *searchbase* definition. If you are running in TSO, set the **LDAP_BASEDN** environment variable using LE runtime environment variable **_CEE_ENVFILE**. See *z/OS: C/C++ Programming Guide*, SC09-4765 for more information. If you are running in the z/OS shell, simply export the **LDAP_BASEDN** environment variable.
-s *scope*	Specify the scope of the search. The *scope* should be one of **base**, **one**, or **sub** to specify a base object, one-level, or subtree search. The default is **sub**.
-a *deref*	Specify how alias dereferencing is done. The *deref* should be one of **never**, **always**, **search**, or **find** to specify that aliases are never dereferenced, always dereferenced, dereferenced when searching, or dereferenced only when locating the base object for the search. The default is to never dereference aliases.
-l *timelimit*	Wait at most *timelimit* seconds for a search to complete. Also note the following: • If a client has passed a limit, then the smaller value of the client value, and the value read from **slapd.conf** will be used. • If the client has not passed a limit, and has bound as the **adminDN**, then the limit will be considered unlimited. • If the client has not passed a limit, and has not bound as the **adminDN**, then the limit will be that which was read from the **slapd.conf** file.
-z *sizelimit*	Limit the results of the search to at most *sizelimit* entries. This makes it possible to place an upper bound on the number of entries that are returned for a search operation. Also note the following: • If a client has passed a limit, then the smaller value of the client value, and the value read from **slapd.conf** will be used. • If the client has not passed a limit, and has bound as the **adminDN**, then the limit will be considered unlimited. • If the client has not passed a limit, and has not bound as the **adminDN**, then the limit will be that which was read from the **slapd.conf** file.
-D *binddn*	Use *binddn* to bind to the LDAP directory. The *binddn* should be a string-represented DN. The default is a NULL string.
-w *bindpasswd*	Use *bindpasswd* as the password for simple authentication. The default is a NULL string.
-h *ldaphost*	Specify the host on which the LDAP server is running. The default is the local host. When the target host is a z/OS LDAP server operating in multi-server mode with dynamic workload management enabled (see the configuring chapter in *z/OS: SecureWay Security Server LDAP Server Administration and Use* for additional information about LDAP server operating modes), the *ldaphost* value should be in the form *group_name.sysplex_domain_name*, where *group_name* is the name of the **sysplexGroupName** identified in the server configuration file and *sysplex_domain_name* is the name or alias of the sysplex domain in which the target server operates.
-p *ldapport*	Specify the TCP port where the LDAP server is listening. The default LDAP port is 389. If not specified and **-Z** is specified, the default LDAP SSL port 636 is used.
-Z	Use a secure SSL connection to communicate with the LDAP server. The **-Z** option is not supported by non-SSL versions of this tool.

Table 7. ldapsearch Options (continued)

Option	Description
-K *keyfile*	Specify the name of the SSL key database file. If the key database file is not in the current directory, specify the fully-qualified key database file name. If a key database file name is not specified, this utility will look for the presence of the **SSL_KEYRING** environment variable with an associated file name. Otherwise, no key database file will be used for server authentication and default trusted certification authority roots will be used. The key database file typically contains one or more certificates of certification authorities (CAs) that are trusted by the client. These types of X.509 certificates are also known as trusted roots. For more information on managing an SSL key database file, see *z/OS: System Secure Sockets Layer Programming* for a description on the use of the **gskkyman** utility to manage the contents of a key database file. Also see the SSL section in *z/OS: SecureWay Security Server LDAP Server Administration and Use* for more information about SSL and certificates. This parameter is ignored if **-Z** is not specified.
-P *keyfilepw*	Specify the key database file password. This password is required to access the encrypted information in the key database file (including the private key). If the key database file does not contain a private key, which is possible on the LDAP client, then the key database file may have been created without a password. In this case, there is no need to specify a password here. This parameter is ignored if **-Z** is not specified.
-N *keyfiledn*	Specify the certificate name in the key database file.

filter
Specify an IETF RFC 1558 compliant LDAP search filter. (See “ldap_search” on page 73 for more information on filters.)

attributes
Specify a space-separated list of attributes to retrieve. If no *attributes* list is given, all are retrieved.

Output Format

If one or more entries are found, each entry is written to standard output in the form:

```
Distinguished Name (DN)
attributename=value
attributename=value
attributename=value
...
```

Multiple entries are separated with a single blank line. If the **-F** option is used to specify a separator character, it will be used instead of the equal sign (=). If the **-t** option is used, the name of a temporary file is used in place of the actual value. If the **-A** option is given, only the `attributename` part is written.

Examples

Following are some **ldapsearch** examples:

- The command:

  ```
  ldapsearch "cn=karen smith" cn telephoneNumber
  ```

  performs a subtree search (using the default search base) for entries with a **commonName** of `karen smith`. The **commonName** and **telephoneNumber** values are retrieved and printed to standard output. The output might look something like this if two entries are found:

```
cn=Karen G Smith, ou=College of Engineering,
ou=Students, ou=People, o=IBM University, c=US
cn=Karen Smith
cn=Karen Grace Smith
cn=Karen G Smith
telephoneNumber=+1 313 555-9489
cn=Karen D Smith, ou=Information Technology Division,
ou=Faculty and Staff,
ou=People, o=IBM University, c=US
cn=Karen Smith
cn=Karen Diane Smith
cn=Karen D Smith
telephoneNumber=+1 313 555-2277
```

- The command:

```
ldapsearch -t "uid=kds" jpegPhoto audio
```

  performs a subtree search using the default searchbase for entries with user ID of `kds`. The **jpegPhoto** and **audio** values are retrieved and written to temporary files. The output might look like this if one entry with one value for each of the requested attributes is found:

```
cn=Karen D Smith, ou=Information Technology Division,
ou=Faculty and Staff,
ou=People, o=IBM University, c=US
audio=/tmp/ldapsearch-audio-a19924
jpegPhoto=/tmp/ldapsearch-jpegPhoto-a19924
```

- The command:

```
ldapsearch -L -s one -b "c=US" "o=university*" o description
```

  performs a one-level search at the `c=US` level for all organizations whose **organizationName** begins with `university`. Search results are displayed in the LDIF format. The **organizationName** and **description** attribute values are retrieved and printed to standard output, resulting in output similar to this:

```
dn: o=University of Alaska Fairbanks, c=US
o: University of Alaska Fairbanks
description: Preparing Alaska for a brave new tomorrow
description: leaf node only
dn: o=University of Colorado at Boulder, c=US
o: University of Colorado at Boulder
description: No personnel information
description: Institution of education and research
dn: o=University of Colorado at Denver, c=US
o: University of Colorado at Denver
o: UCD
o: CU/Denver
o: CU-Denver
description: Institute for Higher Learning and Research
dn: o=University of Florida, c=US
o: University of Florida
o: UFl
description: Shaper of young minds
...
```

- The command:

```
ldapsearch -h ushost -V 3 -M -b "c=US" "objectclass=referral"
```

  performs a subtree search for the `c=US` subtree within the server at host `ushost` (TCP port 389) and returns all referral objects. Note that the search is limited to the single server. No referrals are followed to other servers to find additional referral objects. The output might look something like this if two referral objects are found:

```
o=IBM,c=US
objectclass=referral
ref=ldap://ibmhost:389/o=IBM,c=US
```

```
o=XYZ Company,c=US
objectclass=referral
ref=ldap://XYZhost:390/o=XYZ%20Company,c=US
```

- The command:

  ```
  ldapsearch -h ushost -V 3 -s base -b ""  "objectclass=*"
  ```

  provides the root DSE (DSA-specific entries, where a DSA is a directory server) information for a server. This request can be directed to servers supporting LDAP Version 3 protocol to obtain information about support available in the server. Refer to IETF RFC 2251 *Lightweight Directory Access Protocol (v3)* for a description of the information provided by the server. See *z/OS: SecureWay Security Server LDAP Server Administration and Use* for more information about root DSE and what the z/OS LDAP server returns.

SSL Note

See “SSL Information for LDAP Utilities” on page 86.

Diagnostics

Exit status is 0 if no errors occur. Errors result in a nonzero exit status and a diagnostic message being written to standard error.

ldapsearch

Appendix A. LDAP Header Files

This section contains a description of the header files supplied with the LDAP client. These files are located in the **/usr/lpp/ldapclient/include** directory. To include these files in your applications, enclose the header file name within angle brackets in your source code. For example, to include the **ldap.h** header file, use:

```
#include <ldap.h>
```

lber.h

The **lber.h** header file contains additional definitions for selected LDAP routines. It is included automatically by the **ldap.h** header file. This header defines additional constants, types, and macros that are used with the LDAP APIs.

Figure 2 shows the contents of the **lber.h** header file:

Figure 2. lber.h Header File

```
??=ifdef __COMPILER_VER__
  ??=pragma filetag ("IBM-1047")
??=endif


/*
 * Licensed Materials - Property of IBM
 * 5647-A01
 * (C) Copyright IBM Corp. 1997, 1999
 *"
 */

/*
 * Copyright (c) 1990 Regents of the University of Michigan.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that this notice is preserved and that due credit is given
 * to the University of Michigan at Ann Arbor. The name of the
 * University may not be used to endorse or promote products
 * derived from this software without specific prior written
 * permission.  This software is provided "as is" without express
 * or implied warranty.
 */

#ifndef _LBER_H
#define _LBER_H

/* structure for returning a sequence of octet strings + length */
struct berval {
   unsigned long   bv_len;
   char      *bv_val;
};

typedef struct berelement BerElement;
#define NULLBER   ((BerElement *) 0)

#endif /* _LBER_H */
```

ldap.h

The **ldap.h** header file contains definitions for the LDAP routines. It has the following format:

```
#include <ldap.h>
```

It is a mandatory include file for all applications working with the LDAP APIs. This header defines constants, types, and macros that are used with the interface.

Figure 3 shows the contents of the **ldap.h** header file:

Figure 3. ldap.h Header File

```
??=ifdef __COMPILER_VER__
  ??=pragma filetag ("IBM-1047")
??=endif

/*
 * Licensed Materials - Property of IBM
 * 5647-A01
 * (C) Copyright IBM Corp. 1997, 2000
 *
 */

/*
 * Copyright (c) 1990 Regents of the University of Michigan.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that this notice is preserved and that due credit is given
 * to the University of Michigan at Ann Arbor. The name of the
 * University may not be used to endorse or promote products
 * derived from this software without specific prior written
 * permission.   This software is provided "as is" without express
 * or implied warranty.
 */

#ifndef _LDAP_H
#define _LDAP_H

#ifdef __cplusplus
extern "C" {
#endif

#ifdef _WIN32
#include <winsock.h>
#else
#include <sys/time.h>
#endif

#include <lber.h>

#define LDAP_VERSION2   2
#define LDAP_VERSION3   3
#ifdef LDAPV3
  #define LDAP_VERSION   LDAP_VERSION3
#else
  #define LDAP_VERSION   LDAP_VERSION2
#endif

/* For compatibility w/Netscape implementation of ldap_version(). */
#define LDAP_SECURITY_NONE      0

#define LDAP_PORT    389
#define LDAPS_PORT   636
```

```
#define LDAP_MAX_ATTR_LEN   100

/* possible result types a server can return */
#define LDAP_RES_BIND             0x61L   /* application + constructed */
#define LDAP_RES_SEARCH_ENTRY     0x64L   /* application + constructed */
#define LDAP_RES_SEARCH_RESULT    0x65L   /* application + constructed */
#define LDAP_RES_MODIFY       0x67L   /* application + constructed */
#define LDAP_RES_ADD          0x69L   /* application + constructed */
#define LDAP_RES_DELETE       0x6bL   /* application + constructed */
#define LDAP_RES_MODRDN       0x6dL   /* application + constructed */
#define LDAP_RES_COMPARE      0x6fL   /* application + constructed */
#define LDAP_RES_SEARCH_REFERENCE   0X73L   /* application + constructed */
#define LDAP_RES_EXTENDED         0X78L   /* application + constructed */
#define LDAP_EXTENDED_RES_NAME    0X8aL   /* context specific+primitive*/
#define LDAP_EXTENDED_RES_VALUE   0X8bL   /* context specific+primitive*/
#define LDAP_RES_REFERRAL      0Xa3L   /* context specific+constructed*/
#define LDAP_RES_ANY         (-1L)

/* authentication methods available */
#define LDAP_AUTH_SIMPLE   0x80L   /* context specific + primitive   */
#define LDAP_AUTH_SASL_30  0xa3L
#define LDAP_SASL_SIMPLE   ""

/* search scopes */
#define LDAP_SCOPE_BASE    0x00
#define LDAP_SCOPE_ONELEVEL  0x01
#define LDAP_SCOPE_SUBTREE   0x02

/* bind constants */
#define LDAP_MECHANISM_EXTERNAL   "EXTERNAL"

/* for modifications */
typedef struct ldapmod {
   int      mod_op;
#define LDAP_MOD_ADD       0x00
#define LDAP_MOD_DELETE    0x01
#define LDAP_MOD_REPLACE   0x02
#define LDAP_MOD_BVALUES   0x80
   char      *mod_type;
   union {
      char      **modv_strvals;
      struct berval   **modv_bvals;
   } mod_vals;
#define mod_values   mod_vals.modv_strvals
#define mod_bvalues  mod_vals.modv_bvals
   struct ldapmod   *mod_next;
} LDAPMod;

/*
 * options that can be set/gotten
 */
#define LDAP_OPT_SIZELIMIT      0x00
#define LDAP_OPT_TIMELIMIT      0x01
#define LDAP_OPT_REFERRALS      0x00000002
#define LDAP_OPT_DEREF          0x03
#define LDAP_OPT_RESTART        0x00000004
#define LDAP_OPT_REFHOPLIMIT    0x05
#define LDAP_OPT_DEBUG          0x06

#define LDAP_OPT_SSL_CIPHER      0x07
#define LDAP_OPT_SSL_TIMEOUT     0x08

#define LDAP_OPT_REBIND_FN          0x09
#define LDAP_OPT_SSL                0x0A
#define LDAP_OPT_PROTOCOL_VERSION   0x11
#define LDAP_OPT_SERVER_CONTROLS    0x12
```

ldap.h

```
#define LDAP_OPT_CLIENT_CONTROLS    0x13
#define LDAP_OPT_HOST_NAME          0x30
#define LDAP_OPT_ERROR_NUMBER       0x31
#define LDAP_OPT_ERROR_STRING       0x32
#define LDAP_OPT_EXT_ERROR          0x33

#define LDAP_OPT_UTF8_IO            0xE0
#define LDAP_OPT_SSL_CERTIFICATE_DN 0xE1

#define LDAP_OPT_V2_WIRE_FORMAT    0xE2

#define LDAP_OPT_V2_WIRE_FORMAT_ISO8859_1 0x00
#define LDAP_OPT_V2_WIRE_FORMAT_UTF8      0x01

#define LDAP_OPT_LCS         0x0F

/* option value for no size limit or no time limit on searches */
#define LDAP_NO_LIMIT      0

/* option values for binary options */
#define LDAP_OPT_ON      0x01
#define LDAP_OPT_OFF     0x00

/* option values for dereferencing aliases */
#define LDAP_DEREF_NEVER       0
#define LDAP_DEREF_SEARCHING   1
#define LDAP_DEREF_FINDING     2
#define LDAP_DEREF_ALWAYS      3

/* default limit on nesting of referrals */
#define LDAP_DEFAULT_REFHOPLIMIT   10

/* Debug levels */
#define LDAP_DEBUG_OFF          0x00000000
#define LDAP_DEBUG_TRACE        0x00000001
#define LDAP_DEBUG_PACKETS      0x00000002
#define LDAP_DEBUG_ARGS         0x00000004
#define LDAP_DEBUG_CONNS        0x00000008
#define LDAP_DEBUG_BER          0x00000010
#define LDAP_DEBUG_FILTER       0x00000020
#define LDAP_DEBUG_MESSAGE      0x00000040
#define LDAP_DEBUG_ACL          0x00000080
#define LDAP_DEBUG_STATS        0x00000100
#define LDAP_DEBUG_THREAD       0x00000200
#define LDAP_DEBUG_REPL         0x00000400
#define LDAP_DEBUG_PARSE        0x00000800
#define LDAP_DEBUG_PERFORMANCE  0x00001000
#define LDAP_DEBUG_RDBM         0x00002000
#define LDAP_DEBUG_REFERRAL     0x00004000
#define LDAP_DEBUG_ERROR        0x00008000
#define LDAP_DEBUG_SYSPLEX      0x00010000
#define LDAP_DEBUG_MULTISERVER  0x00020000
#define LDAP_DEBUG_LDAPBE       0x00040000
#define LDAP_DEBUG_STRBUF       0x00080000
#define LDAP_DEBUG_TDBM         0x00100000
#define LDAP_DEBUG_SCHEMA       0x00200000
#define LDAP_DEBUG_ANY          0x7fffffff


/* options for SSL ciphers */
#define LDAP_SSL_RC4_MD5_EX   "03"
#define LDAP_SSL_RC2_MD5_EX   "06"
#define LDAP_SSL_RC4_SHA_US   "05"
#define LDAP_SSL_RC4_MD5_US   "04"
#define LDAP_SSL_DES_SHA_US   "09"
#define LDAP_SSL_3DES_SHA_US  "0A"
```

```
/*
 * possible error codes we can return
 */

#define LDAP_SUCCESS               0x00
#define LDAP_OPERATIONS_ERROR      0x01
#define LDAP_PROTOCOL_ERROR        0x02
#define LDAP_TIMELIMIT_EXCEEDED    0x03
#define LDAP_SIZELIMIT_EXCEEDED    0x04
#define LDAP_COMPARE_FALSE         0x05
#define LDAP_COMPARE_TRUE          0x06
#define LDAP_STRONG_AUTH_NOT_SUPPORTED      0x07
#define LDAP_STRONG_AUTH_REQUIRED  0x08
#define LDAP_PARTIAL_RESULTS       0x09

#define LDAP_REFERRAL              0X0a
#define LDAP_ADMIN_LIMIT_EXCEEDED  0X0b
#define LDAP_UNAVAILABLE_CRITICAL_EXTENSION   0X0c
#define LDAP_CONFIDENTIALITY_REQUIRED         0x0d
#define LDAP_SASLBIND_IN_PROGRESS             0x0e

#define LDAP_NO_SUCH_ATTRIBUTE           0x10
#define LDAP_UNDEFINED_TYPE              0x11
#define LDAP_INAPPROPRIATE_MATCHING      0x12
#define LDAP_CONSTRAINT_VIOLATION        0x13
#define LDAP_TYPE_OR_VALUE_EXISTS        0x14
#define LDAP_INVALID_SYNTAX              0x15

#define LDAP_NO_SUCH_OBJECT         0x20
#define LDAP_ALIAS_PROBLEM          0x21
#define LDAP_INVALID_DN_SYNTAX      0x22
#define LDAP_IS_LEAF                0x23
#define LDAP_ALIAS_DEREF_PROBLEM    0x24

#define LDAP_INAPPROPRIATE_AUTH     0x30
#define LDAP_INVALID_CREDENTIALS    0x31
#define LDAP_INSUFFICIENT_ACCESS    0x32
#define LDAP_BUSY                   0x33
#define LDAP_UNAVAILABLE            0x34
#define LDAP_UNWILLING_TO_PERFORM   0x35
#define LDAP_LOOP_DETECT            0x36

#define LDAP_NAMING_VIOLATION       0x40
#define LDAP_OBJECT_CLASS_VIOLATION 0x41
#define LDAP_NOT_ALLOWED_ON_NONLEAF 0x42
#define LDAP_NOT_ALLOWED_ON_RDN     0x43
#define LDAP_ALREADY_EXISTS         0x44
#define LDAP_NO_OBJECT_CLASS_MODS   0x45
#define LDAP_RESULTS_TOO_LARGE      0x46

#define LDAP_AFFECTS_MULTIPLE_DSAS  0X47

#define LDAP_OTHER               0x50
#define LDAP_SERVER_DOWN         0x51
#define LDAP_LOCAL_ERROR         0x52
#define LDAP_ENCODING_ERROR      0x53
#define LDAP_DECODING_ERROR      0x54
#define LDAP_TIMEOUT             0x55
#define LDAP_AUTH_UNKNOWN        0x56
#define LDAP_FILTER_ERROR        0x57
#define LDAP_USER_CANCELLED      0x58
#define LDAP_PARAM_ERROR         0x59
#define LDAP_NO_MEMORY           0x5a
#define LDAP_CONNECT_ERROR       0x5b
#define LDAP_NOT_SUPPORTED       0x5c
#define LDAP_CONTROL_NOT_FOUND   0x5d
#define LDAP_NO_RESULTS_RETURNED 0x5e
```

ldap.h

```
#define LDAP_MORE_RESULTS_TO_RETURN  0x5f

#define LDAP_URL_ERR_NOTLDAP     0x60
#define LDAP_URL_ERR_NODN        0x61
#define LDAP_URL_ERR_BADSCOPE    0x62
#define LDAP_URL_ERR_MEM         0x63

#define LDAP_CLIENT_LOOP         0x64
#define LDAP_REFERRAL_LIMIT_EXCEEDED   0x65

#define LDAP_SSL_ALREADY_INITIALIZED       0x70
#define LDAP_SSL_INITIALIZE_FAILED         0x71
#define LDAP_SSL_CLIENT_INIT_NOT_CALLED    0x72
#define LDAP_SSL_PARAM_ERROR               0x73
#define LDAP_SSL_HANDSHAKE_FAILED          0x74
#define LDAP_SSL_GET_CIPHER_FAILED         0x75
#define LDAP_SSL_NOT_AVAILABLE             0x76

#define LDAP_NO_EXPLICIT_OWNER         0x80
#define LDAP_NO_EXPLICIT_ACL           0x81

/*
 * This structure represents both ldap messages and ldap responses.
 * These are really the same, except in the case of search responses,
 * where a response has multiple messages.
 */

typedef struct ldapmsg LDAPMessage;
#define NULLMSG ((LDAPMessage *) NULL)

/*

 * structure representing an ldap connection
 */
typedef struct ldap  LDAP;

/*
 * type for ldap_set_rebind_proc()
 */
typedef int (*LDAPRebindProc)( struct ldap *ld, char **dnp,
         char **passwdp, int *authmethodp, int freeit );

/*
 * types for ldap URL handling
 */
typedef struct ldap_url_desc {
    char   *lud_host;
    int    lud_port;
    char   *lud_dn;
    char   **lud_attrs;
    int    lud_scope;
    char   *lud_filter;
    char   *lud_string;   /* for internal use only */
} LDAPURLDesc;
#define NULLLDAPURLDESC ((LDAPURLDesc *)NULL)

typedef struct _LDAPVersion {
    int sdk_version;
    int protocol_version;
    int SSL_version;
    int security_level;
    char ssl_max_cipher[ 65 ] ;
    char ssl_min_cipher[ 65 ] ;
} LDAPVersion;

typedef struct _LDAPControl {
    char *ldctl_oid;
```

```
    struct berval ldctl_value;
    int           ldctl_iscritical;
} LDAPControl;

/* Function prototypes */
#ifndef _NO_PROTO
#define LDAP_P(x) x
#else
#define LDAP_P(x) ()
#endif

int ldap_abandon ( LDAP *ld, int msgid );
int ldap_abandon_ext ( LDAP *ld, int msgid,
   LDAPControl **serverctrls,
   LDAPControl **clientctrls );
int ldap_add ( LDAP *ld, char *dn, LDAPMod **attrs );
int ldap_add_s ( LDAP *ld, char *dn, LDAPMod **attrs );
int ldap_add_ext ( LDAP *ld, char *dn, LDAPMod **attrs,
   LDAPControl **serverctrls,
   LDAPControl **clientctrls,
   int *msgidp );
int ldap_add_ext_s ( LDAP *ld, char *dn, LDAPMod **attrs,
   LDAPControl **serverctrls,
   LDAPControl **clientctrls );
int ldap_bind ( LDAP *ld, char *who, char *passwd,
   int authmethod );
int ldap_bind_s ( LDAP *ld, char *who, char *cred,
   int method );
int ldap_simple_bind ( LDAP *ld, char *who, char *passwd );
int ldap_simple_bind_s ( LDAP *ld, char *who, char *passwd );
void ldap_set_rebind_proc ( LDAP *ld,
   LDAPRebindProc rebindproc );
int ldap_compare ( LDAP *ld, char *dn, char *attr,
   char *value );
int ldap_compare_s ( LDAP *ld, char *dn, char *attr,
   char *value );
int ldap_compare_ext ( LDAP *ld, char *dn, char *attr,
   struct berval *bvalue,
   LDAPControl **serverctrls,
   LDAPControl **clientctrls,
   int *msgidp );
int ldap_compare_ext_s ( LDAP *ld, char *dn, char *attr,
   struct berval *bvalue,
   LDAPControl **serverctrls,
   LDAPControl **clientctrls );
int ldap_delete ( LDAP *ld, char *dn );
int ldap_delete_s ( LDAP *ld, char *dn );
int ldap_delete_ext ( LDAP *ld, char *dn,
   LDAPControl **serverctrls,
   LDAPControl **clientctrls,
   int *msgidp );
int ldap_delete_ext_s ( LDAP * ld, char *dn,
   LDAPControl **serverctrls,
   LDAPControl **clientctrls );
int ldap_result2error ( LDAP *ld, LDAPMessage *r, int freeit );
char *ldap_err2string ( int err );
void ldap_perror ( LDAP *ld, char *s );
int ldap_get_errno ( LDAP *ld );
int ldap_modify ( LDAP *ld, char *dn, LDAPMod **mods );
int ldap_modify_s ( LDAP *ld, char *dn, LDAPMod **mods );
int ldap_modify_ext ( LDAP *ld, char *dn, LDAPMod **mods,
   LDAPControl **serverctrls,
   LDAPControl **clientctrls,
   int *msgidp );
int ldap_modify_ext_s ( LDAP *ld, char *dn, LDAPMod **mods,
   LDAPControl **serverctrls,
   LDAPControl **clientctrls );
```

ldap.h

```
int ldap_modrdn ( LDAP *ld, char *dn, char *newrdn,
   int deleteoldrdn );
int ldap_modrdn_s ( LDAP *ld, char *dn, char *newrdn,
   int deleteoldrdn );
LDAP *ldap_open ( char *host, int port );
LDAP *ldap_init ( char *defhost, int defport );
int ldap_set_option ( LDAP *ld, int optionToSet,
   void *optionValue );
int ldap_set_option_np ( LDAP *ld, int optionToSet, ... );
int ldap_get_option ( LDAP *ld, int optionToGet,
   void *optionValue );
int ldap_version ( LDAPVersion *version );
LDAPMessage *ldap_first_entry ( LDAP *ld, LDAPMessage *chain );
LDAPMessage *ldap_next_entry ( LDAP *ld, LDAPMessage *entry );
int ldap_count_entries ( LDAP *ld, LDAPMessage *chain );
int ldap_get_entry_controls_np( LDAP *ld, LDAPMessage *entry,
   LDAPControl ***serverctrlsp );
LDAPMessage *ldap_first_message ( LDAP *ld, LDAPMessage *chain );
LDAPMessage *ldap_next_message  ( LDAP *ld, LDAPMessage *chain );
int ldap_count_messages ( LDAP *ld, LDAPMessage *chain );
LDAPMessage *ldap_first_reference ( LDAP *ld, LDAPMessage *res );
LDAPMessage *ldap_next_reference ( LDAP *ld, LDAPMessage *res );
int ldap_count_references ( LDAP *ld, LDAPMessage *result );
int ldap_parse_reference_np( LDAP *ld, LDAPMessage *ref, char ***referralsp,
   LDAPControl ***serverctrlsp, int freeit );
char *ldap_get_dn ( LDAP *ld, LDAPMessage *entry );
char **ldap_explode_dn ( char *dn, int notypes );
char **ldap_explode_rdn ( char *rdn, int notypes );
char *ldap_dn2ufn ( char *dn );
char *ldap_first_attribute ( LDAP *ld, LDAPMessage *entry,
   BerElement **ber );
char *ldap_next_attribute ( LDAP *ld, LDAPMessage *entry,
   BerElement *ber );
int ldap_count_attributes ( LDAP *ld, LDAPMessage *entry );
char **ldap_get_values ( LDAP *ld, LDAPMessage *entry,
   char *target );
struct berval **ldap_get_values_len ( LDAP *ld,
   LDAPMessage *entry, char *target );
int ldap_count_values ( char **vals );
int ldap_count_values_len ( struct berval **vals );
void ldap_value_free ( char **vals );
void ldap_value_free_len ( struct berval **vals );
int ldap_result ( LDAP *ld, int msgid, int all,
   struct timeval *timeout, LDAPMessage **result );
int ldap_msgfree ( LDAPMessage *lm );
int ldap_msgid ( LDAPMessage *res );
int ldap_msgtype ( LDAPMessage *res );
int ldap_search ( LDAP *ld, char *base, int scope, char *filter,
   char **attrs, int attrsonly );
int ldap_search_s ( LDAP *ld, char *base, int scope,
   char *filter, char **attrs, int attrsonly,
   LDAPMessage **res );
int ldap_search_st ( LDAP *ld, char *base, int scope,
   char *filter, char **attrs, int attrsonly,
   struct timeval *timeout, LDAPMessage **res );
int ldap_search_ext ( LDAP *ld, char *base, int scope, char *filter,
   char **attrs, int attrsonly,
   LDAPControl **serverctrls,
   LDAPControl **clientctrls,
   struct timeval *timeoutp,
   int sizelimit, int *msgidp );
int ldap_search_ext_s ( LDAP *ld, char *base, int scope, char *filter,
   char **attrs, int attrsonly,
   LDAPControl **serverctrls,
   LDAPControl **clientctrls,
   struct timeval *timeoutp,
   int sizelimit,
```

```
   LDAPMessage **res );
int ldap_unbind ( LDAP *ld );
int ldap_unbind_s ( LDAP *ld );
void ldap_mods_free ( LDAPMod **mods, int freemods );
void ldap_control_free ( LDAPControl *ctrl );
void ldap_controls_free ( LDAPControl **ctrls );
void ldap_memfree ( char *mem );
int ldap_is_ldap_url ( char *url );
int ldap_url_parse ( char *url, LDAPURLDesc **ludpp );
void ldap_free_urldesc ( LDAPURLDesc *ludp );
int ldap_url_search ( LDAP *ld, char *url, int attrsonly );
int ldap_url_search_s ( LDAP *ld, char *url, int attrsonly,
   LDAPMessage **res );
int ldap_url_search_st ( LDAP *ld, char *url, int attrsonly,
   struct timeval *timeout, LDAPMessage **res );

int   ldap_set_cipher( LDAP *ld, char *userString );

int ldap_ssl_start ( LDAP *ld, char *keyfile, char *keyfile_pw,
   char *keyfile_dn );

int ldap_ssl_client_init ( char *keyfile, char *keyfile_pw,
   int sslTimeout, int *pSSLReasonCode );
LDAP *ldap_ssl_init ( char *host, int port, char *keyfile_dn );

int ldap_sasl_bind ( LDAP *ld, char *dn, char *mechansim,
   struct berval *credentials,
   LDAPControl **serverctrls,
   LDAPControl **clientctrls,
   int* msgidp );
int ldap_sasl_bind_s ( LDAP *ld, char* dn, char *mechansim,
   struct berval *credentials,
   LDAPControl **serverctrls,
   LDAPControl **clientctrls,
   struct berval **servercredp );
int ldap_rename ( LDAP* ld, char *dn, char *newdn, char *newparent,
   int deleteoldrdn,
   LDAPControl **serverctrls, LDAPControl **clientctrls,
   int *msgidp );
int ldap_rename_s ( LDAP* ld,char *dn, char *newdn, char *newparent,
   int deleteoldrdn,
   LDAPControl **serverctrls, LDAPControl **clientctrls );
int ldap_parse_result ( LDAP* ld, LDAPMessage *result, int *errcodep,
   char **matcheddnp, char **errmsgp,
   char ***referralsp, LDAPControl ***serverctrlsp,
   int freeint );
int ldap_parse_sasl_bind_result ( LDAP* ld, LDAPMessage *result,
      struct berval **servercredp,
   int freeit );
int ldap_parse_extended_result(LDAP *ld, LDAPMessage *res,
   char **resultoidp, struct berval **resultdata, int freeit);

#ifdef __cplusplus
}
#endif

#endif /* _LDAP_H */
```

ldapssl.h

The **ldapssl.h** header file contains definitions for the LDAP SSL routines. It is an include for all applications working with the LDAP SSL APIs. This header defines constants that are used with this interface.

Figure 4 shows the contents of the **ldapssl.h** header file:

Figure 4. ldapssl.h Header File

```
??=ifdef __COMPILER_VER__
  ??=pragma filetag ("IBM-1047")
??=endif


/*
 * Licensed Materials - Property of IBM
 * 5647-A01
 * (C) Copyright IBM Corp. 1997, 1999
 *
 */

#ifndef _LDAPSSL_H
#define _LDAPSSL_H

/*
   Return values returned from ldap_ssl_client_init(), ldap_ssl_init()
   and ldap_ssl_start()
*/

#define LDAP_SSL_INITIALIZE_OK            0 /* Successful Completion        */
#define LDAP_SSL_KEYFILE_IO_ERROR         1 /* Attention: Keyring io error  */
#define LDAP_SSL_KEYFILE_OPEN_FAILED      2 /* Attention: Keyring open error*/
#define LDAP_SSL_KEYFILE_BAD_FORMAT       3 /* Attention: Keyring format bad*/
#define LDAP_SSL_KEYFILE_BAD_PASSWORD     4 /* Attention: Keyring pw bad    */
#define LDAP_SSL_KEYFILE_BAD_MALLOC       5 /* Error: Malloc failed         */
#define LDAP_SSL_KEYFILE_NOTHING_TO_WRITE 6
#define LDAP_SSL_KEYFILE_WRITE_FAILED     7
#define LDAP_SSL_KEYFILE_NOT_FOUND        8
#define LDAP_SSL_KEYFILE_BAD_DNAME        9 /* Error: Distinguished name bad*/
#define LDAP_SSL_KEYFILE_BAD_KEY         10
#define LDAP_SSL_KEYFILE_KEY_EXISTS      11
#define LDAP_SSL_KEYFILE_BAD_LABEL       12
#define LDAP_SSL_KEYFILE_DUPLICATE_NAME  13
#define LDAP_SSL_KEYFILE_DUPLICATE_KEY   14
#define LDAP_SSL_KEYFILE_DUPLICATE_LABEL 15
#define LDAP_SSL_ERR_INIT_PARM_NOT_VALID 100 /* Error: Cipher spec bad      */
#define LDAP_SSL_INIT_HARD_RT            101 /* Attention: No keyring file
                                                 or password      */
#define LDAP_SSL_INIT_SEC_TYPE_NOT_VALID 102 /* Error: Security type bad    */
#define LDAP_SSL_INIT_V2_TIMEOUT_NOT_VALID 103 /* Error&colon.V2 timeout value bad*/
#define LDAP_SSL_INIT_V3_TIMEOUT_NOT_VALID 104 /* Error&colon.V3 timeout value bad*/
#define LDAP_SSL_KEYFILE_CERT_EXPIRED      105 /* Error:Certificate expired */


/*
   Return codes. These are returned as an LDAP_OPT_EXTERROR, using
   ldap_get_option(), when an SSL-related error has occurred.

   Use ldap_get_option() with LDAP_OPT_EXTERROR to get a more detailed SSL
        error code whenever LDAP_SSL_HANDSHAKE_FAILED is returned from an
        LDAP call
        to use the new #defines listed above.
*/
```

```
#define LDAP_SSL_SOC_BAD_V2_CIPHER                      -40
#define LDAP_SSL_SOC_BAD_V3_CIPHER                      -41
#define LDAP_SSL_SOC_BAD_SEC_TYPE                       -42
#define LDAP_SSL_SOC_BAD_SEC_TYPE_COMBINATION           -102
#define LDAP_SSL_SOC_NO_READ_FUNCTION                   -43
#define LDAP_SSL_SOC_NO_WRITE_FUNCTION                  -44

#define LDAP_SSL_ERROR_NO_CIPHERS                       -1
#define LDAP_SSL_ERROR_NO_CERTIFICATE                   -2
#define LDAP_SSL_ERROR_BAD_CERTIFICATE                  -4
#define LDAP_SSL_ERROR_UNSUPPORTED_CERTIFICATE_TYPE     -6
#define LDAP_SSL_ERROR_IO                               -10
#define LDAP_SSL_ERROR_BAD_MESSAGE                      -11
#define LDAP_SSL_ERROR_BAD_MAC                          -12
#define LDAP_SSL_ERROR_UNSUPPORTED                      -13
#define LDAP_SSL_ERROR_BAD_CERT_SIG                     -14
#define LDAP_SSL_ERROR_BAD_CERT                         -15
#define LDAP_SSL_ERROR_BAD_PEER                         -16
#define LDAP_SSL_ERROR_PERMISSION_DENIED                -17
#define LDAP_SSL_ERROR_SELF_SIGNED                      -18
#define LDAP_SSL_ERROR_BAD_MALLOC                       -20
#define LDAP_SSL_ERROR_BAD_STATE                        -21    /* V3 */
#define LDAP_SSL_ERROR_SOCKET_CLOSED                    -22
#define LDAP_SSL_ERROR_LDAP_SSL_INITIALIZATION_FAILED   -23
#define LDAP_SSL_ERROR_HANDLE_CREATION_FAILED           -24
#define LDAP_SSL_ERROR_UNKNOWN_ERROR                    -99

#endif /* _LDAPSSL_H */
```

ldapssl.h

Appendix B. Sample Makefile

Following is a sample Makefile.

Figure 5. Sample Makefile

```
#  THIS FILE CONTAINS SAMPLE CODE.  IBM PROVIDES THIS CODE ON AN
#  'AS IS' BASIS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS
#  OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
#  OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
#
CFLAGS = -W0,DLL -Dmvs -D_OPEN_THREADS -DMVS_PTHREADS -D_ALL_SOURCE -DEBCDIC_PLATFORM -D_LONGMAP
CFLAGS +=-I/usr/include -I.
SIDEFILE=/usr/lib/GLDCLDAP.x
LIBS = $(SIDEFILE)
OBJS2 = line64.o
MODS = ldapsearch ldapdelete ldapmodify ldapmodrdn sdelete ldapadd
default:  $(MODS)
ldapsearch: ldapsearch.o $(OBJS2)
   c89 -o ldapsearch ldapsearch.o $(OBJS2) $(LIBS)
ldapdelete: ldapdelete.o
   c89 -o ldapdelete ldapdelete.o $(LIBS)
ldapmodify: ldapmodify.o $(OBJS2)
   c89 -o ldapmodify ldapmodify.o $(OBJS2) $(LIBS)
ldapmodrdn: ldapmodrdn.o
   c89 -o ldapmodrdn ldapmodrdn.o $(LIBS)
sdelete: sdelete.o
   c89 -o sdelete sdelete.o $(LIBS)
ldapadd: ldapmodify
   ln -s ./ldapmodify ldapadd
clean:
   rm -f *.o
clobber: clean
   rm -f $(MODS)
```

Makefile

Appendix C. Example Programs

This appendix shows two sample programs that use the LDAP programming interface.

The ldapdelete.c Example Program

The following example program (found in the **/usr/lpp/ldap/examples** directory) shows how the LDAP programming interface can be used to interact with a Directory Service. This program can be used to delete an entry from the Directory.

Figure 6. ldapdelete.c Example Program

```
??=ifdef __COMPILER_VER__
??=pragma filetag ("IBM-1047")
??=endif

/*********************************************************************/
/* THIS FILE CONTAINS SAMPLE CODE.  IBM PROVIDES THIS CODE ON AN     */
/* 'AS IS' BASIS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS        */
/* OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES  */
/* OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.           */
/*********************************************************************/

/*
 * Copyright (c) 1995 Regents of the University of Michigan.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that this notice is preserved and that due credit is given
 * to the University of Michigan at Ann Arbor. The name of the University
 * may not be used to endorse or promote products derived from this
 * software without specific prior written permission. This software
 * is provided "as is" without express or implied warranty.
 */

/* ldapdelete.c - simple program to delete an entry using LDAP */

#include <stdio.h>
#include <string.h>
#include <strings.h>
#include <stdlib.h>
#include <ctype.h>
#include <ldap.h>
#include <locale.h>

#ifndef TRUE
    #define TRUE 1
#endif

#ifndef FALSE
    #define FALSE 0
#endif

static LDAP *ld;
static char *prog;
static char *binddn = NULL;
static char *passwd = NULL;
static char *ldaphost = "localhost";
static int  ldapport = LDAP_PORT;
static int  not = FALSE;
static int  verbose = FALSE;
static int  contoper = FALSE;
```

ldapdelete.c

```
static int  follow_referrals = LDAP_OPT_ON;
static int  deref = LDAP_DEREF_NEVER;
static int  ldapversion = LDAP_VERSION2;
static int  manageDsa = FALSE;
static LDAPControl manageDsaIT = {
                "2.16.840.1.113730.3.4.2",   /*OID*/
                { 0, NULL },                 /*no value*/
                LDAP_OPT_ON                  /*critical*/
            };
static LDAPControl *M_controls[2] = { &manageDsaIT, NULL};

static void usage( char *s );
static int  dodelete( LDAP *ld, char *dn );
int rebindproc( LDAP *ld, char **dnp, char **pwp, int *methodp, int freeit );


main( int argc, char **argv )
{
    char    *optpattern = "nvRMZc?h:V:p:D:w:d:f:K:P:N:";
    int     ssl = FALSE;
    char    *keyfile = NULL, *keyfile_pw = NULL, *keyfile_dn = NULL;
    char    *p, buf[ 4096 ];
    FILE    * fp;
    int     i, rc=LDAP_SUCCESS, port = FALSE;
    int     debugLevel = 0;
    int     debugSpecified = FALSE;
    int     failureReasonCode ;
    extern  char *optarg;
    extern  int  optind;

    setlocale( LC_ALL, "" );

    if ( prog = strrchr( argv[0], '/' ) ) {  /* Strip off any path info
                                              * on program name
                                              */
        ++prog;
    }
    else {
        prog = argv[0];
    }

    not = verbose = contoper = ssl = port = FALSE;
    fp = NULL;

   while ( ( i = getopt( argc, argv, optpattern ) ) != EOF ) {
        switch ( i ) {
        case 'V':
            ldapversion = atoi( optarg );
            if ( ldapversion != LDAP_VERSION2 &&
                 ldapversion != LDAP_VERSION3 ) {
                fprintf( stderr, "Incorrect version level supplied.\n");
                fprintf( stderr, "Supported values for the -V parameter"
                                 " are 2 and 3\n");
                exit( 1 );
            }
            break ;
        case 'c':   /* continue even if error encountered */
            contoper = TRUE;
            break;
        case 'h':   /* ldap host */
            ldaphost = strdup( optarg );
            break;
        case 'D':   /* bind DN */
            binddn = strdup( optarg );
            break;
        case 'w':   /* password */
            passwd = strdup( optarg );
```

```
        break;
    case 'f':   /* read DNs from a file */
        if ( ( fp = fopen( optarg, "r" ) ) == NULL ) {
            perror( optarg );
            exit( 1 );
        }
        break;
    case 'd':
        debugLevel = atoi( optarg );
        debugSpecified = TRUE;
        break;
    case 'p':
        ldapport = atoi( optarg );
        port = TRUE;
        break;
    case 'n':   /* print deletes, don't actually do them */
        not = TRUE;
        break;
    case 'R':   /* don't automatically chase referrals */
        follow_referrals  = LDAP_OPT_OFF;
        break;
    case 'M':
        manageDsa = TRUE;
        break;
    case 'v':   /* verbose mode */
        verbose = TRUE;
        break;
    case 'K':
        keyfile = strdup( optarg );
        break;
    case 'P':
        keyfile_pw = strdup( optarg );
        break;
    case 'N':
        keyfile_dn = strdup( optarg );
        break;
    case 'Z':
        ssl = TRUE;
        break;
    case '?':
    default:
        usage( prog );
        exit( 1 );
    }
}

if ( manageDsa && ( ldapversion == LDAP_VERSION2 ) ) {
    fprintf( stderr, "-M option requires version 3.\n" );
    exit(1);
}

if ( fp == NULL ) {
    if ( optind >= argc ) {
        fp = stdin;
    }
}

if ( !not ) {
    if ( ssl )  {
        if ( !port ) {
            ldapport = LDAPS_PORT;
        }

        if ( keyfile == NULL ) {
            keyfile = getenv( "SSL_KEYRING" );
            if ( keyfile != NULL ) {
                keyfile = strdup(keyfile);
```

```
            }
        }

        if (verbose) {
            printf( "ldap_ssl_client_init( %s, %s, 0,"
                    " &failureReasonCode )\n",
                    keyfile ? keyfile : "NULL",
                    keyfile_pw ? keyfile_pw : "NULL" );
        }
        rc = ldap_ssl_client_init( keyfile, keyfile_pw, 0,
                                   &failureReasonCode ) ;
        if ( rc != LDAP_SUCCESS ) {
            fprintf( stderr,
                     "ldap_ssl_client_init failed! rc == %d,"
                     " failureReasonCode == %d\n",
                     rc, failureReasonCode );
            exit( 1 ) ;
        }
        if ( verbose ) {
            printf( "ldap_ssl_init( %s, %d, %s )\n",
                    ldaphost, ldapport,
                    keyfile_dn ? keyfile_dn : "NULL" );
        }
        ld = ldap_ssl_init( ldaphost, ldapport, keyfile_dn ) ;
        if ( ld == NULL ) {
            fprintf( stderr, "ldap_ssl_init failed\n" ) ;
            perror( ldaphost ) ;
            exit( 1 ) ;
        }
    }
    else {
        if ( verbose ) {
            printf( "ldap_init(%s, %d) \n", ldaphost, ldapport );
        }
        if ( ( ld = ldap_init( ldaphost, ldapport ) ) == NULL ) {
            perror( ldaphost );
            exit( 1 );
        }
    }

    ldap_set_option_np( ld, LDAP_OPT_PROTOCOL_VERSION, ldapversion );
    if ( debugSpecified ) {
       ldap_set_option_np( ld, LDAP_OPT_DEBUG, debugLevel );
    }
    ldap_set_option_np( ld, LDAP_OPT_DEREF, deref );
    ldap_set_option_np( ld, LDAP_OPT_REFERRALS, follow_referrals );

    if ( binddn != NULL ) {
        ldap_set_rebind_proc( ld, (LDAPRebindProc)rebindproc );
    }

    if ( ldapversion == LDAP_VERSION2 && binddn != NULL ) {
       /*
        * Bind is required for LDAP V2 protocol,
        * but not for V3 (or later) protocols.
        * We also bind if a bind DN was specified.
        */
       if ( ldap_bind_s( ld, binddn, passwd, LDAP_AUTH_SIMPLE )
            != LDAP_SUCCESS ) {
           ldap_perror( ld, "ldap_bind" );
           exit( 1 );
       }
    }
} /* ! not */

if ( fp == NULL ) {
    for ( ; (rc == LDAP_SUCCESS  contoper) && optind < argc; ++optind ) {
```

```
            rc = dodelete( ld, argv[ optind ] );
        }
    }
    else {
        rc = LDAP_SUCCESS;
        while ( (rc == LDAP_SUCCESS  contoper) &&
                fgets( buf, sizeof(buf), fp ) != NULL ) {
            buf[ strlen( buf ) - 1 ] = '\0';    /* remove trailing newline */
            if ( *buf != '\0' ) {
                rc = dodelete( ld, buf );
            }
        }
    }

    if ( !not ) {
        ldap_unbind( ld );
    }

    exit( rc );
}

static void usage( char *s )
{
    fprintf( stderr, "usage: %s [options] [ -f file [ < entryfile ] dn ... ]\n"
                  , s );
    fprintf( stderr, "where:\n" );
    fprintf( stderr, "    dn\tdistinguished name of entry to delete\n" );
    fprintf( stderr, "    entryfile\tfile containing DNs to delete\n" );
    fprintf( stderr, "    \t\ton consecutive lines\n" );
    fprintf( stderr, "options:\n" );
    fprintf( stderr, "    -?\t\tprint this text\n" );
    fprintf( stderr, "    -V version\tselect LDAP protocol version"
                                        " (2 or 3; default is 2)\n");
    fprintf( stderr, "    -c\t\tcontinue even if error encountered\n");
    fprintf( stderr, "    -n\t\tshow what would be done but don't actually"
                                        " delete\n" );
    fprintf( stderr, "    -v\t\trun in verbose mode (diagnostics to"
                                        " standard output)\n" );
    fprintf( stderr, "    -R\t\tdo not automatically follow referrals\n" );
    fprintf( stderr, "    -M\t\tTreat referral objects as normal entries."
                                        " (requires -V 3)\n" );
    fprintf( stderr, "    -d level\tset LDAP debugging level to 'level'\n" );
    fprintf( stderr, "    -f file\tperform sequence of deletes listed"
                                        " in 'file'\n" );
    fprintf( stderr, "    -D binddn\tbind dn\n" );
    fprintf( stderr, "    -w passwd\tbind passwd (for simple"
                                        " authentication)\n" );
    fprintf( stderr, "    -h host\tldap server\n" );
    fprintf( stderr, "    -p port\tport on ldap server\n" );
    fprintf( stderr, "    -Z\t\tuse a secure ldap connection for the"
                                        " operation\n");
    fprintf( stderr, "    -K keyfile\tfile to use for keys/certificates\n");
    fprintf( stderr, "    -P key_pw\tkeyfile password\n");
    fprintf( stderr, "    -N key_dn\tCertificate Name in keyfile\n");
}

static int  dodelete( LDAP *ld, char *dn )
{
    int rc;

    if ( verbose ) {
        printf( "%sdeleting entry %s\n", not ? "!" : "", dn );
    }
    if ( not ) {
        rc = LDAP_SUCCESS;
    }
    else {
```

ldapdelete.c

```
        rc = ldap_delete_ext_s( ld, dn,
                                manageDsa ? M_controls : NULL,
                                NULL );
        if ( rc != LDAP_SUCCESS ) {
            ldap_perror( ld, "ldap_delete" );
        }
        else if ( verbose ) {
            printf( "entry removed\n" );
        }
    }

    return ( rc );
}


int rebindproc( LDAP *ld, char **dnp, char **pwp, int *methodp,
                int freeit )
{
    if ( !freeit ) {
        *methodp = LDAP_AUTH_SIMPLE;
        if ( binddn != NULL ) {
            *dnp = strdup( binddn );
            *pwp = strdup( passwd );
        }
        else {
            *dnp = NULL;
            *pwp = NULL;
        }
    }
    else {
        free( *dnp );
        free( *pwp );
    }
    return ( LDAP_SUCCESS );
}
```

The ldapsearch.c Example Program

The following program is an example of searching entries using the LDAP APIs. The example program can also be found in the **/usr/lpp/ldap/examples** directory.

Note the following regarding the **ldapsearch.c** example program and all program source shipped in **/usr/lpp/ldap/examples**:

- The example source code as shipped with the LDAP Server is only compilable from the z/OS shell environment. As shipped, the code is not compilable from the batch environment.
- If compilation from a batch environment is required, compilation flags and libraries required can be found in the Makefile. See “Using TSO and Batch Jobs” on page 7 for more information about linking, compiling, and running LDAP client applications using TSO and batch jobs.
- Be aware that there are lines in the example code that exceed 80 characters in length. If the modules are placed into datasets, the datasets must be allocated such that these lines are not truncated.
- See *z/OS: UNIX System Services Command Reference*, SA22-7802 for more details about running the **c89** program from the z/OS shell and from batch.

Figure 7. ldapsearch.c Example Program

```
??=ifdef __COMPILER_VER__
??=pragma filetag ("IBM-1047")
??=endif



/*********************************************************************/
/* THIS FILE CONTAINS SAMPLE CODE.  IBM PROVIDES THIS CODE ON AN     */
/* 'AS IS' BASIS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS        */
/* OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES  */
/* OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.           */
/*********************************************************************/

/*
 * Copyright (c) 1995 Regents of the University of Michigan.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that this notice is preserved and that due credit is given
 * to the University of Michigan at Ann Arbor. The name of the University
 * may not be used to endorse or promote products derived from this
 * software without specific prior written permission. This software
 * is provided "as is" without express or implied warranty.
 */

/* ldapsearch.c - simple program to search, list, or read entries
 *                using LDAP
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <strings.h>
#include <ctype.h>
#include <ldap.h>
#include <line64.h>

#include <locale.h>

#ifndef TRUE
    #define TRUE 1
#endif
```

ldapsearch.c

```
#ifndef FALSE
    #define FALSE 0
#endif

#define DEFSEP      "="

static int  rebindproc( LDAP *ld, char **dnp, char **pwp, int *methodp, int freeit );
static int  dosearch( LDAP *, char *, int, char **, int, char *, char *);
static void print_entry( LDAP *, LDAPMessage *, int);
static int  write_ldif_value( char *, char *, unsigned long );
static void usage( char *s );
static int  write_ldif_value_or_bvalue( char *, char *, unsigned long, char *, unsigned long );

static char *prog = NULL;
static char *binddn = NULL;
static char *passwd = NULL;
static char *base = NULL;
static char *ldaphost = "localhost";
static int  ldapport = LDAP_PORT;
static char *sep = DEFSEP;
static int  verbose, not, allow_binary, vals2tmp, ldif;
static int  ldapversion = LDAP_VERSION2;


main( int argc, char **argv )
{
    char *infile, *filtpattern, **attrs, line[ BUFSIZ ];
char *optpattern = "ZnvtRMABLD:V:s:f:h:b:d:p:F:a:w:l:z:S:K:P:N:";
    FILE * fp;
    int  rc, i, first, scope, deref, attrsonly, port = 0;
    int  timelimit, sizelimit, authmethod;
    int  follow_referrals;
    LDAP * ld;
    extern char *optarg;
    extern int  optind;
    int  debugLevel = 0;
    int  debugSpecified = FALSE;
    int  ssl = FALSE;
    char *keyfile = NULL, *keyfile_pw = NULL, *keyfile_dn = NULL;
    int  failureReasonCode;
    FILE * cf_fd;
    char *mechanism = "EXTERNAL";
    int  sasl_bind = FALSE;
    struct berval **servercred = NULL;
    int  manageDsa = FALSE;
    LDAPControl manageDsaIT = { "2.16.840.1.113730.3.4.2",   /*OID*/
        {0, NULL},                  /*no value*/
        LDAP_OPT_ON                 /*critical*/
    };
    LDAPControl *M_controls[2] = { NULL, NULL};

    M_controls[0] = &manageDsaIT;

    if (prog = strrchr(argv[0], '/'))   /* Strip off any path info
                                         * on program name
                                         */
        ++prog;
    else
        prog = argv[0];

    setlocale(LC_ALL, "");

    infile = NULL;
```

```
deref = verbose = allow_binary = not = vals2tmp = attrsonly = ldif = 0;
follow_referrals = LDAP_OPT_ON;        /* default to chase referrals */
sizelimit = timelimit = 0;
scope = LDAP_SCOPE_SUBTREE;

while (( i = getopt( argc, argv, optpattern )) != EOF ) {

    switch ( i ) {
    case 'V':   /* use version 3 functions */
        ldapversion = atoi(optarg);
        if ( (ldapversion != LDAP_VERSION2) &&
             (ldapversion != LDAP_VERSION3) ) {
            fprintf(stderr, "Incorrect LDAP protocol version selected.\n");
            fprintf(stderr, "Supported values are 2 and 3\n");
            usage( prog );
            exit( 1 );
        }
        break;
 case 'S':   /* use Sasl Bind functions */
    if ( strncasecmp( optarg, "external", 8 ) == 0 ) {
            sasl_bind = TRUE;
   }
   else {
  fprintf( stderr, "only supported mechanism is EXTERNAL\n" );
  usage( prog );
            exit( 1 );
   }
   break;
    case 'n':   /* do Not do any searches */
        not = TRUE;
        break;
    case 'v':   /* verbose mode */
        verbose = TRUE;
        break;
    case 'd':
        debugLevel = atoi( optarg );
        debugSpecified = TRUE;
        break;
    case 't':   /* write attribute values to /tmp files */
        vals2tmp = TRUE;
        break;
    case 'R':   /* don't automatically chase referrals */
        follow_referrals = LDAP_OPT_OFF;
        break;
    case 'M':       /* manage referral objects as normal entries */
        manageDsa = TRUE;
        break;
    case 'A':   /* retrieve attribute names only -- no values */
        attrsonly = TRUE;
        break;
    case 'L':   /* print entries in LDIF format */
        ldif = TRUE;
        /* fall through -- always allow binary when outputting LDIF */
    case 'B':   /* allow binary values to be printed */
        allow_binary = TRUE;
        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( prog );
            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( prog );
            exit( 1 );
        }
        break;

    case 'F':   /* field separator */
        sep = strdup( optarg );
        break;
    case 'f':   /* input file */
        infile = strdup( optarg );
        break;
    case 'h':   /* ldap host */
        ldaphost = strdup( optarg );
        break;
    case 'b':   /* searchbase */
        base = strdup( optarg );
        break;
    case 'D':   /* bind DN */
        binddn = strdup( optarg );
        break;
    case 'p':   /* ldap port */
        ldapport = atoi( optarg );
        port = 1;
        break;
    case 'w':   /* bind password */
        passwd = strdup( optarg );
        break;
    case 'l':   /* time limit */
        timelimit = atoi( optarg );
        break;
    case 'z':   /* size limit */
        sizelimit = atoi( optarg );
        break;
```

```
    case 'K':
        keyfile = strdup( optarg );
        break;
    case 'P':
        keyfile_pw = strdup( optarg );
        break;
    case 'Z':
        ssl = TRUE;
        break;
    case 'N':
        keyfile_dn = strdup( optarg );
        break;
    default:
        usage( prog );
        exit( 1 );
    }
}

if ( manageDsa && (ldapversion == LDAP_VERSION2)) {
    fprintf( stderr, "-M option requires version 3. -M ignored.\n");
}

if (( base == NULL )) {
    base = getenv( "LDAP_BASEDN" );
    if (base != NULL) {
        base = strdup(base);
    }
    /* if NULL will start at top */
}

if ( argc - optind < 1 ) {
    usage( prog );
}
filtpattern = strdup( argv[ optind ] );
if ( argv[ optind + 1 ] == NULL ) {
    attrs = NULL;
}
else {
    attrs = &argv[ optind + 1 ];
}
if ( infile != NULL ) {
    if ( infile[0] == '-' && infile[1] == '\0' ) {
        fp = stdin;
    }
    else if (( fp = fopen( infile, "r" )) == NULL ) {
        perror( infile );
        exit( 1 );
    }
}

if ( !not ) {
    if (ssl) {
        if (!port) {
            ldapport = LDAPS_PORT;
        }


        if ( keyfile == NULL ) {
            keyfile = getenv("SSL_KEYRING");
            if (keyfile != NULL) {
                keyfile = strdup(keyfile);
```

```
            }
        }

        if (verbose) {
            printf( "ldap_ssl_client_init( %s, %s, 0,"
                                        " &failureReasonCode )\n",
                    ((keyfile) ? keyfile : "NULL"),
                    ((keyfile_pw) ? keyfile_pw : "NULL"));
        }
        rc = ldap_ssl_client_init( keyfile, keyfile_pw, 0,
                                   &failureReasonCode ) ;
        if (rc != LDAP_SUCCESS) {
            fprintf( stderr,
                     "ldap_ssl_client_init failed! rc == %d,"
                                        " failureReasonCode == %d\n",
                     rc, failureReasonCode );
            exit( 1 ) ;
        }
        if (verbose) {
            printf("ldap_ssl_init( %s, %d, %s )\n", ldaphost, ldapport,
                   ((keyfile_dn) ? keyfile_dn : "NULL"));
        }
        ld = ldap_ssl_init( ldaphost, ldapport, keyfile_dn ) ;
        if (ld == NULL) {
            fprintf( stderr, "ldap_ssl_init failed\n" ) ;
            perror( ldaphost ) ;
            exit( 1 ) ;
        }
    }
    else {
        if (verbose) {
            printf("ldap_init(%s, %d) \n", ldaphost, ldapport);
        }
        if ((ld = ldap_init(ldaphost, ldapport)) == NULL) {
            perror(ldaphost);
            exit(1);
        }
    }

    ldap_set_option_np(ld, LDAP_OPT_PROTOCOL_VERSION, ldapversion);
    if ( debugSpecified ) {
       ldap_set_option_np(ld, LDAP_OPT_DEBUG, debugLevel);
    }
    ldap_set_option_np(ld, LDAP_OPT_DEREF, deref);
    ldap_set_option_np(ld, LDAP_OPT_REFERRALS, follow_referrals);
    ldap_set_option_np( ld, LDAP_OPT_TIMELIMIT, timelimit);
    ldap_set_option_np( ld, LDAP_OPT_SIZELIMIT, sizelimit);
    if ( manageDsa ) {
        ldap_set_option_np( ld, LDAP_OPT_SERVER_CONTROLS, M_controls);
    }

    if ( binddn != NULL ) {
        ldap_set_rebind_proc( ld, (LDAPRebindProc)rebindproc );
    }

    if ( ldapversion != LDAP_VERSION3
         (ldapversion == LDAP_VERSION3 && binddn != NULL
          && sasl_bind == FALSE) ) {
       /*
        * When running LDAP Version 3 protocol, bind only if
        * a bind DN was specified.
```

```
            */
           authmethod = LDAP_AUTH_SIMPLE;
           if ( ldap_bind_s( ld, binddn, passwd, authmethod )
                                                != LDAP_SUCCESS ) {
               ldap_perror( ld, "ldap_bind" );
               exit( 1 );
           }
       }
   else if ( ldapversion == LDAP_VERSION3 && sasl_bind == TRUE ) {
       if ( ldap_sasl_bind_s(ld, NULL, mechanism, NULL, NULL, NULL,
               servercred) != LDAP_SUCCESS ) {
         ldap_perror( ld, "ldap_sasl_bind_s" );
         exit( 1 );
      }
   }
    } /* ! not */

    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, NULL );
    }
    else {
        rc = LDAP_SUCCESS;
        first = 1;
        while ( rc == LDAP_SUCCESS &&
                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 );
        }
    }

    if ( !not ) {
        ldap_set_option_np( ld, LDAP_OPT_SERVER_CONTROLS, NULL);
        ldap_unbind( ld );
    }

    exit( rc );
}

static void usage( char *s )
```

ldapsearch.c

```
{
    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, "    -?\t\tprint this text\n" );
    fprintf( stderr, "    -V version\tselect LDAP protocol version"
                                        " (2 or 3; default is 2)\n");
    fprintf( stderr, "    -S mechanism select SASL bind mechanism"
                               " (only supported mechanism is EXTERNAL)\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, "    -A\t\tretrieve attribute names only (no values)\n" );
    fprintf( stderr, "    -B\t\tdo not suppress printing of non-printable"
                                        " values\n" );
    fprintf( stderr, "    -L\t\tprint entries in LDIF format"
                                        " (-B is implied)\n" );
    fprintf( stderr, "    -R\t\tdo not automatically follow referrals\n" );
    fprintf( stderr, "    -M\t\tManage referral objects as normal entries."
                                        " (requires -V 3)\n" );
    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, "    -f file\tperform sequence of searches listed in"
                                       " 'file'. ('-' implies stdin)\n" );
    fprintf( stderr, "    -b basedn\tbase dn for search. LDAP_BASEDN in"
                                       " environment is default\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" );
    fprintf( stderr, "    -h host\tldap server\n" );
    fprintf( stderr, "    -p port\tport on ldap server\n" );
    fprintf( stderr, "    -Z\t\tuse a secure ldap connection for search\n");
    fprintf( stderr, "    -K keyfile\tfile to use for keys/certificates\n");
    fprintf( stderr, "    -P key_pw\t keyfile password\n");
    fprintf( stderr, "    -N key_dn\t Certificate Name in keyfile\n");
}

static int  dosearch( LDAP *ld, char *base, int scope, char **attrs,
                      int attrsonly, char *filtpatt, char *value )
{
    char filter[ BUFSIZ ], **val;
    int  rc, first, matches;
    int  references;
    char **referrals = NULL;
    int  errcode;
    char *matched, *errmsg;
    LDAPMessage * res, *e;
    int  msgidp;
```

```
if ( value ) {
    sprintf( filter, filtpatt, value );
}
else {
    strncpy ( filter, filtpatt, BUFSIZ - 1 );
}

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 ( ldap_get_errno( ld ) );
}

matches = 0;
references = 0;
first = 1;

for ( ; ; ) {
    rc = ldap_result( ld, LDAP_RES_ANY, 0, NULL, &res );
    if ( rc == 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 );
    }
    else if ( rc == LDAP_RES_SEARCH_REFERENCE ) {
        references++;
        /* parse and free the search reference */
        ldap_parse_reference_np( ld, res, &referrals, NULL, 1 );
       if ( referrals != NULL ) {
            int i;
            for ( i = 0; referrals[i] != NULL; i++) {
                fprintf( stderr,
                         (i == 0) ? "Unfollowed search reference: %s\n" :
                         "                              %s\n",
                         referrals[i]);
            }
            fflush( stderr );
            ldap_value_free( referrals );
            referrals = NULL;
        }
    }
    else {
        /* must be a search result */
        break;
    }
} /* end for */
```

```
    if ( rc == -1 ) {
        ldap_perror( ld, "ldap_result" );
        return ( rc );
    }

    if (ldapversion > LDAP_VERSION2) {
        if ( ( rc = ldap_parse_result( ld, res, &errcode, &matched, &errmsg,
                                    &referrals, NULL, 1 ) )
                                                != LDAP_SUCCESS ) {
            fprintf( stderr, "ldap_search: error parsing result: %d, %s\n",
                     rc, ldap_err2string( rc ) );
        }
        else {
            if ( errcode != LDAP_SUCCESS ) {
                fprintf( stderr, "ldap_search: %s\n",
                                 ldap_err2string( errcode ) );
                if ( matched != NULL ) {
                    if ( *matched != '\0' )
                        fprintf( stderr, "ldap_search: matched: %s\n",
                                         matched );
                    ldap_memfree( matched );
                }
                if ( errmsg != NULL ) {
                    if ( *errmsg != '\0' )
                        fprintf( stderr, "ldap_search: additional info: %s\n",
                                         errmsg );
                    ldap_memfree( errmsg );
                }
            }
            if ( referrals != NULL ) {
                int i;
                for ( i = 0; referrals[i] != NULL; i++) {
                    fprintf( stderr, "%s %s\n",
                             (i == 0) ? "Unfollowed referral:" :
                             "                 ",
                             referrals[i]);
                }
                ldap_value_free( referrals );
                referrals = NULL;
            }
        }
        fflush( stderr );
    }
    else {
        if (( rc = ldap_result2error( ld, res, 1 )) != LDAP_SUCCESS ) {
            ldap_perror( ld, "ldap_search" );
        }
    }

    if ( verbose ) {
        printf( "%d matches\n", matches );
        if (references > 0) {
            printf( "%d unfollowed references\n", references );
        }
    }

    return ( rc );
}


static void print_entry( LDAP *ld, LDAPMessage *entry, int attrsonly)
```

```
{
    char    *a, *dn, tmpfname[ 64 ];
    int i, j, printable = TRUE;
    BerElement      * ber;
    struct berval **bvals;
    FILE        * tmpfp;
    char    **vals = NULL;

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

    for ( a = ldap_first_attribute( ld, entry, &ber ); a != NULL;
        a = ldap_next_attribute( ld, entry, ber ) ) {
        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 ) {
            vals = ldap_get_values( ld, entry, a);
            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 {

                    int value_len = bvals[ i ]->bv_len;
                    char    *str_value = vals[ i ];
                    if ( ldif ) {
                        write_ldif_value_or_bvalue( a,
```

```
                                                str_value,
                                                value_len,
                                                bvals[ i ]->bv_val,
                                                value_len );
                        }
                        else {
                            printable = TRUE;
                            if (strlen(str_value) == value_len) {
                                for ( j = 0; j < value_len; j++) {
                                    if ( !isprint( str_value[ j ] )) {
                                        printable = FALSE;
                                        break;
                                    }
                                }
                            }

                            printf( "%s%s%s\n", a, sep,
                                    printable ? str_value :
                                    (allow_binary ? bvals[ i ]->bv_val :
                                     "NOT Printable") );
                        }
                    }
                }
                ldap_value_free_len( bvals );
                ldap_value_free( vals );
            }
            ldap_memfree( a );
        }
    }


static int
write_ldif_value_or_bvalue( char *type, char *value, unsigned long vallen,
                            char *bvalue, unsigned long bvallen)
{
    char    *ldif;

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

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

    return ( 0 );
}


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

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

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

```
    return ( 0 );
}


static int  rebindproc( LDAP *ld, char **dnp, char **pwp, int *methodp,
                        int freeit )
{
    if ( !freeit ) {
        *methodp = LDAP_AUTH_SIMPLE;
        if ( binddn != NULL ) {
            *dnp = strdup( binddn );
            *pwp = strdup ( passwd );
        }
        else {
            *dnp = NULL;
            *pwp = NULL;
        }
    }
    else {
        free ( *dnp );
        free ( *pwp );
    }
    return ( LDAP_SUCCESS );
}
```

ldapsearch.c

Appendix D. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION ″AS IS″ WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300

2455 South Road
Poughkeepsie, NY 12601-5400
U.S.A

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Programming Interface Information

This *z/OS: SecureWay Security Server LDAP Client Programming* documents intended Programming Interfaces that allow the customer to write programs to obtain services of z/OS LDAP.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	IBMLink	OS/390	SecureWay
AIX/6000	Library Reader	RACF	z/OS
BookManager	OS/2	Resource Link	

Java and all Java-based trademarks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product or service names may be the trademarks or service marks of others.

Glossary

This glossary defines technical terms and abbreviations used in z/OS LDAP documentation. If you do not find the term you are looking for, refer to the index of the appropriate z/OS manual or view IBM Glossary of Computing Terms, located at:

`http://www.ibm.com/ibm/terminology`

This glossary includes terms and definitions from:

- *IBM Dictionary of Computing*, SC20-1699.
- *Information Technology—Portable Operating System Interface (POSIX),* from the POSIX series of standards for applications and user interfaces to open systems, copyrighted by the Institute of Electrical and Electronics Engineers (IEEE).
- *American National Standard Dictionary for Information Systems,* ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036. Definitions are identified by the symbol (A) after the definition.
- *Information Technology Vocabulary*, developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1.SC1).
- *CCITT Sixth Plenary Assembly Orange Book, Terms and Definitions* and working documents published by the International Telecommunication Union, Geneva, 1978.
- Open Software Foundation (OSF).

A

API. Application program interface.

application program interface (API). A functional interface supplied by the operating system or by a separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or the licensed program.

attribute. Information of a particular type concerning an object and appearing in an entry that describes the object in the directory information base (DIB). It denotes the attribute's type and a sequence of one or more attribute values, each accompanied by an integer denoting the value's syntax.

B

binding. A relationship between a client and a server involved in a remote procedure call.

C

CDS. Cell Directory Service.

Cell Directory Service (CDS). A DCE component. A distributed replicated database service that stores names and attributes of resources located in a cell. CDS manages a database of information about the resources in a group of machines called a DCE cell.

certificate. Used to prove your identity. A secure server must have a certificate and a public-private key pair. A certificate is issued and signed by a Certificate Authority (CA).

client. A computer or process that accesses the data, services, or resources of another computer or process on the network. Contrast with *server*.

cipher. A method of transforming text in order to conceal its meaning.

D

data hierarchy. A data structure consisting of sets and subsets such that every subset of a set is of lower rank than the data of the set.

data model. (1) A logical view of the organization of data in a database. (2) In a database, the user's logical view of the data in contrast to the physically stored data, or storage structure. (3) A description of the organization of data in a manner that reflects information structure of an enterprise.

database. A collection of data with a given structure for accepting, storing, and providing, on demand, data for multiple users.

DCE. Distributed Computing Environment.

directory. (1) A logical unit for storing entries under one name (the directory name) in a CDS namespace. Each physical instance of a directory is called a replica. (2) A collection of open systems that cooperates to hold a logical database of information about a set of objects in the real world.

directory schema. The set of rules and constraints concerning directory information tree (DIT) structure, object class definitions, attribute types, and syntaxes that characterize the directory information base (DIB).

directory service. The directory service is a central repository for information about resources in a distributed system.

distinguished name (DN). One of the names of an object, formed from the sequence of RDNs of its object entry and each of its superior entries.

Distributed Computing Environment (DCE). A comprehensive, integrated set of services that supports the development, use, and maintenance of distributed applications. DCE is independent of the operating system and network; it provides interoperability and portability across heterogeneous platforms.

DN. Distinguished name.

E

environmental variable. A variable included in the current software environment that is available to any called program that requests it.

L

LDAP. Lightweight Directory Access Protocol.

Lightweight Directory Access Protocol (LDAP). A client/server protocol for accessing a directory service.

O

object class. An identified family of objects that share certain characteristics. An object class can be specific to one application or shared among a group of applications. An application interprets and uses an entry's class-specific attributes based on the class of the object that the entry describes.

P

private key. Used for the encryption of data. A secure server keeps its private key secret. A secure server sends clients its public key so they can encrypt data to the server. The server then decrypts the data with its private key.

programming interface. The supported method through which customer programs request software services. The programming interface consists of a set of callable services provided with the product.

protocol. A set of semantic and syntactic rules that determines the behavior of functional units in achieving communication.

public key. Used for the encryption of data. A secure server makes its public key widely available so that its clients can encrypt data to send to the server. The server then decrypts the data with its private key.

R

RDN. Relative distinguished name.

referral. An outcome that can be returned by a directory system agent that cannot perform an operation itself. The referral identifies one or more other directory system agents more able to perform the operation.

relative distinguished name (RDN). A component of a DN. It identifies an entry distinctly from any other entries which have the same parent.

S

SASL. Simple Authentication Security Layer.

schema. See *directory schema*.

Secure Sockets Layer (SSL) security. A facility used to protect LDAP access.

server. On a network, the computer that contains programs, data, or provides the facilities that other computers on the network can access. Contrast with *client*.

Simple Authentication Security Layer (SASL). Refers to a method of binding using SSL authentication and the client's certificate identity.

SSL. Secure Sockets Layer

T

thread. A single sequential flow of control within a process.

X

X.500. The CCITT/ISO standard for the open systems interconnection (OSI) application-layer directory. It allows users to register, store, search, and retrieve information about any objects or resources in a network or distributed system.

X/OPEN Directory Service (XDS). An application program interface that DCE uses to access its directory service components. XDS provides facilities for adding, deleting, and looking up names and their attributes. The XDS library detects the format of the name to be looked up and directs the calls it receives to either GDS or CDS. XDS uses the X/OPEN object management (XOM) API to define and manage its information.

X/OPEN object management (XOM). An interface for creating, deleting, and accessing objects containing information. It is an object-oriented architecture: each object belongs to a particular class, and classes can be derived from other classes inheriting the characteristics of the original classes. The representation of the object

is transparent to the programmer; the object can be manipulated only through the XOM interface.

XOM. The X/OPEN Object Management API.

Bibliography

This bibliography provides a list of publications that are useful when using the LDAP programming interface. The complete title, order number, and a brief description is given for each publication.

IBM C/C++ Language Publication

- *z/OS: C/C++ Programming Guide*, SC09-4765

  This book describes how to develop applications in the C/C++ language in z/OS.

IBM z/OS SecureWay Security Server Publication

- *z/OS: SecureWay Security Server LDAP Server Administration and Use*, SC24-5923

  This book describes how to install, configure, and run the LDAP server. It is intended for administrators who will maintain the server and database.

IBM z/OS Cryptographic Services Publication

- *z/OS: System Secure Sockets Layer Programming*, SC24-5901

  Contains guidance and reference information that an application programmer needs to use the System Secure Sockets Layer (SSL) callable programming interfaces. SSL is a communications layer that provides data privacy and integrity, as well as server and client authentication based on public key certificates.

Index

F

G

H

I

J

L

M

N

O

P

R

S

T

U

W

X

Z

Readers' Comments — We'd Like to Hear from You

z/OS
SecureWay® Security Server
LDAP Client Programming

Publication No. SC24-5924-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

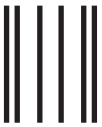
Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SC24-5924-00

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Dept. G60
1701 North Street
Endicott NY 13760-5553

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

Program Number: 5694-A01

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC24-5924-00