z/OS

# Security Server LDAP Client Programming

SC24-5924-02

z/OS

# Security Server
# LDAP Client Programming

SC24-5924-02

**Note**

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 185.

**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.

**Third Edition, September 2002**

This is a major revision of SC24-5924-01.

This edition applies to Version 1 Release 4 of z/OS (5694-A01), to Version 1 Release 4 of z/OS.e (5655-G52), and to all subsequent releases and modifications until otherwise indicated in new editions.

IBM® welcomes your comments. A form for readers’ comments may be provided at the back of this document, or you may address your comments to the following address:

International Business Machines Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX (United States & Canada): 1+845+432-9405
FAX (Other Countries):
Your International Access Code +1+845+432-9405

IBMLink™ (United States customers only): IBMUSM10(MHVRCFS)
Internet e-mail: mhvrcfs@us.ibm.com
World Wide Web: http://www.ibm.com/servers/eserver/zseries/zos/webqs.html

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 document
- 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** . . . vii

**About this document** . . . ix
Who should use this document . . . ix
How this document is organized . . . ix
Conventions used in this document . . . ix
Where to find more information . . . ix
  Softcopy publications . . . x
  z/OS online library . . . x
  Accessing licensed documents on the Web . . . x
  Using LookAt to look up message explanations . . . x

**Summary of changes** . . . xiii

**Chapter 1. LDAP programming** . . . 1
How LDAP is defined . . . 2
Data model . . . 2
LDAP names . . . 3
Function overview . . . 3
Using the socksified client . . . 5
Compiling, linking, and running a program . . . 7
  Using TSO and batch jobs . . . 8
Using the API . . . 9
  Basic structure . . . 9
Performing an operation . . . 10
  Example: adding an entry . . . 10
  Example: modifying an entry . . . 11
  Example: deleting an entire entry . . . 11
  Example: changing the RDN of an entry and relocating the entry . . . 12
  Example: comparing an attribute value with its value in an entry in the directory . . . 12
  Example: reading a directory entry’s contents . . . 12
| Example: listing all objectClass attribute values for all entries directly below a given entry . . . 13
  Example: reading all objectClass attribute values for all entries below a given entry . . . 13
Getting results . . . 13
Error processing . . . 14
  Using ldap_get_errno and ldap_parse_result . . . 14
  Using ldap_err2string and ldap_get_option . . . 15
Tracing . . . 15
Thread safety . . . 17
Client-side search results caching . . . 17
Synchronous versus asynchronous operation . . . 17
Calling the LDAP APIs from other languages . . . 17
LDAP client for Java . . . 17

**Chapter 2. LDAP routines** . . . 19
LDAP controls . . . 24
Session controls . . . 24
Supported client controls . . . 25
Using RACF® data . . . 27
ldap_abandon . . . 28
ldap_add . . . 30
ldap_bind . . . 32
ldap_compare . . . 37

ldap_delete . . . . . 39
ldap_enetwork_domain . . . . . 41
ldap_error . . . . . 42
ldap_extended_operation . . . . . 46
ldap_first_attribute . . . . . 48
ldap_first_entry/reference . . . . . 50
ldap_get_dn . . . . . 53
ldap_get_values . . . . . 54
ldap_init . . . . . 57
ldap_memcache . . . . . 70
ldap_memfree . . . . . 73
ldap_message . . . . . 74
ldap_modify . . . . . 76
ldap_parse_result . . . . . 79
ldap_rename . . . . . 82
ldap_result . . . . . 85
ldap_search . . . . . 87
ldap_server . . . . . 92
ldap_ssl . . . . . 107
ldap_url . . . . . 112

**Chapter 3. LDAP operation utilities** . . . . . 115
Running the LDAP operation utilities in the z/OS shell . . . . . 115
Running the LDAP operation utilities in TSO . . . . . 115
Using the command line utilities . . . . . 116
SSL/TLS information for LDAP utilities . . . . . 116
  Using RACF key rings . . . . . 117
  CRAM-MD5 authentication to an IBM Directory Server . . . . . 117
ldapdelete utility . . . . . 118
ldapmodify and ldapadd utilities . . . . . 121
ldapmodrdn utility . . . . . 131
ldapsearch utility . . . . . 135

**Appendix A. LDAP header files** . . . . . 141
lber.h . . . . . 141
ldap.h . . . . . 142
ldapssl.h . . . . . 154

**Appendix B. Sample Makefile** . . . . . 159

**Appendix C. Example programs** . . . . . 161
The ldapdelete.c example program . . . . . 161
The ldapsearch.c example program . . . . . 169

**Appendix D. Accessibility** . . . . . 183
Using assistive technologies . . . . . 183
Keyboard navigation of the user interface . . . . . 183

**Notices** . . . . . 185
Programming interface information . . . . . 186
Trademarks . . . . . 186

**Bibliography** . . . . . 187

**Glossary** . . . . . 189

**Index** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 193

# Tables

1. LDAP API functions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
2. Debug levels . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
3. LDAP error codes and descriptions . . . . . . . . . . . . . . . . . . . . . . 43
4. The optionValue parameter specifications. . . . . . . . . . . . . . . . . . . . 58
5. ldapdelete options . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 118
6. ldapmodify and ldapadd options . . . . . . . . . . . . . . . . . . . . . . . 121
7. ldapmodrdn options . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 131
8. ldapsearch options. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 135

# About this document

This document supports z/OS (5694-A01) and z/OS.e (5665-G52) and describes the Lightweight Directory Access Protocol (LDAP) client application development for z/OS Security Server.

## Who should use this document

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

## How this document is organized

This document 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 19 describes each LDAP client routine.
- Chapter 3, “LDAP operation utilities” on page 115 describes the LDAP operation utilities and how to run them.
- Appendix A, “LDAP header files” on page 141 shows each LDAP header file.
- Appendix B, “Sample Makefile” on page 159 shows a sample Makefile.
- Appendix C, “Example programs” on page 161 shows examples of how to use the LDAP programming interface.

## Conventions used in this document

This document 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 as 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 document references information in other documents. For complete titles and order numbers of the documents 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 documents that are useful for z/OS LDAP, see "Bibliography" on page 187.

## 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 documents 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.

## 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 document 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 documents on the Web

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

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

Licensed documents are available only to customers with a z/OS license. Access to these documents requires an IBM Resource Link user ID and password, and a key code. With your z/OS order you received a *Memo to Licensees*, (GI10-0671), that includes this key code.

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

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

To register for access to the z/OS licensed documents:

1. Sign in to Resource Link using your Resource Link user ID and password.
2. Select **User Profiles** located on the left-hand navigation bar.

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

Printed licensed documents are not available from IBM.

You can use the PDF format on either *z/OS Licensed Product Library CD-ROM* or IBM Resource Link to print licensed documents.

## Using LookAt to look up message explanations

LookAt is an online facility that allows you to look up explanations for most messages you encounter, as well as for some system abends and codes. Using LookAt to find information is faster than a conventional search because in most cases LookAt goes directly to the message explanation.

You can access LookAt from the Internet at:

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

or from anywhere in z/OS where you can access a TSO/E command line (for example, TSO/E prompt, ISPF, z/OS UNIX System Services running OMVS). You can also download code from the *z/OS: Collection* and the LookAt Web site that will allow you to access LookAt from a handheld computer (Palm Pilot VIIx suggested).

To use LookAt as a TSO/E command, you must have LookAt installed on your host system. You can obtain the LookAt code for TSO/E from a disk on your *z/OS: Collection* or from the **News** section on the LookAt Web site.

Some messages have information in more than one document. For those messages, LookAt displays a list of documents in which the message appears.

# Summary of changes

**Summary of changes**
**for SC24-5924-02**
**z/OS Version 1 Release 4**

This book contains information previously presented in *z/OS Security Server LDAP Client Programming*, SC24-5924-01, which supports z/OS Version 1 Release 2 and z/OS Version 1 Release 3.

## New Information

- The **ldapmodrdn** utility has been updated with full support for Modify DN operations as defined in RFC 2251 - *Lightweight Directory Access Protocol (v3)*, including rename of non-leaf nodes and subtree relocation. In addition, support has been added for new controls to specify a time limit for the Modify DN operation and to request DN realignment as part of the operation.
- The LDAP operation utilities have all been updated to support CRAM-MD5 (Challenge Response Authentication Mechanism - RFC 2104) and DIGEST-MD5 (RFC 2931) authentication bind mechanisms. In order to support these new bind mechanisms, a new client control has been added to specify the authentication identity. An additional client control has been added to specify the realm name to be used during DIGEST-MD5 authentication.
- The LDAP SASL bind APIs are updated to support CRAM-MD5 and DIGEST-MD5 mechanism.
- Updated the process for specifying debug levels for LDAP operation utilities.
- LDAP operation utilities updated to support specifying a System SSL key ring stash file.
- The **ldap_set_option** and **ldap_set_option_np** APIs are updated for a new option to specify the debug values as a key word string.
- The **ldap_ssl_client_init** API supports specifying a System SSL key ring stash file.
- New cipher values can be specified for the **LDAP_OPT_SSL_CIPHER** option on the **ldap_set_option** and **ldap_set_option_np** APIs.
- The LDAP client APIs support both Secure Sockets Layer (SSL) Version 3 and Transport Layer Security (TLS) Version 1 for secure protected sessions.
- An appendix with z/OS product accessibility information has been added.
- Information is added to indicate this document supports z/OS.e.

## Changed Information

- Updated the **ldap.h** header file.
- Updated the **ldapssl.h** header file.
- The default version of the LDAP operation utilities has been updated from LDAP Version 2 to LDAP Version 3.

## Removed Information

- The IBM JNDI provider (**ibmjndi.jar**) is no longer shipped and supported. Migration to the Sun JNDI provider is required.

This document includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

You may notice changes in the style and structure of some content in this book—for example, headings that use uppercase for the first letter of initial words only, and procedures that have a different look and format. The changes are ongoing improvements to the consistency and retrievability of information in our books.

## Summary of changes for SC24-5924-01 z/OS Version 1 Release 2

This book contains information previously presented in *z/OS Security Server LDAP Client Programming*, SC24-5924-00, which supports z/OS Version 1 Release 1.

The following summarizes the changes to that information:

### New information

- Added the following new APIs:
  - **ldap_enetwork_domain_get**
  - **ldap_enetwork_domain_set**
  - **ldap_extended_operation**
  - **ldap_extended_operation_s**
  - **ldap_memcache_destroy**
  - **ldap_memcache_flush**
  - **ldap_memcache_get**
  - **ldap_memcache_init**
  - **ldap_memcache_set**
  - **ldap_memcache_update**
  - **ldap_parse_extended_result**
  - **ldap_server_conf_save**
  - **ldap_server_free_list**
  - **ldap_server_locate**
- Added information on global cache support.
- Added information on the **ibm-serverHandledSearchRequest** client control.
- Added information for Kerberos Version 5 bind.

### Changed Information

- Updated the **ldap.h** header file.
- Updated the process for specifying debug levels.

### Moved Information

- The **ldap_unbind** and **ldap_unbind_s** APIs are now located within the **ldap_init** group of APIs.
- The list of deprecated LDAP APIs are now located at the beginning of the LDAP routines chapter.

This document includes terminology, maintenance, and editorial changes, including changes to improve consistency and retrievability.

# 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 System Services to store and extract information into and from a Directory Service. The LDAP server, a component of the z/OS Security Server, can be used to store and extract information on z/OS using the LDAP protocol. See *z/OS: Security Server LDAP Server Administration and Use* for more information.

| Regarding security, five authentication methods are supported: simple authentication, certificate
| authentication, Kerberos credentials authentication, CRAM-MD5 authentication, and DIGEST-MD5
| 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.

| Secure Socket Layer (SSL) or Transport Layer Security (TLS) can be used to secure the socket
| connection between the client and the server by encrypting the data transferred over the connection. TLS
| is based upon SSL V3. Through a protocol handshake between the client and server, the choice of TLS or
| SSL is decided with TLS being the preferred protocol. In the case of a simple bind, the encryption protects
| the password.

With certificate authentication, the identity from the client certificate sent to the LDAP server on an SSL/TLS 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.

With Kerberos credentials authentication, a client application and an LDAP server accepting Kerberos authentication mutually authenticate each other using a Key Distribution Center (KDC). The identity is determined by algorithms on the server. Kerberos authentication is also referred to as “SASL GSS API bind” and is provided by the **ldap_sasl_bind** API.

| With CRAM-MD5 and DIGEST-MD5 authentication, authentication is accomplished in a series of
| challenges and responses between the client application and server. The response from the client
| application to the server has a hashed password that is calculated by using an algorithm that is known by
| both the client application and server. The server checks to make certain that the authentication is correct
| by calculating its own password hash and comparing it to the client calculated password hash.
| CRAM-MD5 and DIGEST-MD5 authentication 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). IETF RFC 2251 *Lightweight Directory Access Protocol (v3)* defines the LDAP Version 3 specification. LDAP Version 3 is what is implemented by the LDAP client interfaces for z/OS. Version 3 of this protocol is extended by the following RFCs:

RFC 2251 - *Lightweight Directory Access Protocol (v3)*

RFC 2253 - *Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names*

RFC 2255 - *The LDAP URL Format*

| RFC 2829 - *Authentication Methods for LDAP*

| RFC 2830 - *Lightweight Directory Access Protocol (v3): Extension for Transport Layer Security*

| RFC 2831 - *Using Digest Authentication as a SASL Mechanism*

| RFC 2849 - *The LDAP Data Interchange Format (LDIF) - Technical Specification*

| 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.

IETF RFC 1823 *The LDAP Application Programming Interface* 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.

Other RFCs of interest include:
- RFC 1738 - *Uniform Resource Locators (URL)*
- RFC 1777 - *Lightweight Directory Access Protocol*
- RFC 1778 - *The String Representation of Standard Attribute Syntaxes*
- RFC 1779 - *A String Representation of Distinguished Names*
- RFC 1959 - *An LDAP URL Format*
- RFC 1960 - *A String Representation of LDAP Search Filters*
- RFC 2052 - *A DNS RR for Specifying the Location of Services (DNS SRV)*
- | RFC 2195 - *IMAP/POP AUTHorize Extension for Simple Challenge/Response*
- | RFC 2222 - *Simple Authentication and Security Layer (SASL)*

## 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 **Directory String** and **Octet String**.

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* 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. Furthermore, attribute types are provided to the program as null-terminated character strings instead of object identifiers.

## LDAP names

The LDAP protocol and APIs use “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 concatenating 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 *A String Representation of Distinguished Names* 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. **GLDDHUTD** is also loaded. Refer to “Compiling, linking, and running a program” on page 7 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 symbolic link in **/usr/lib** points to an external
| link for **GLDCLDAP** defined in **/usr/lpp/ldapclient/lib**.

The LDAP API consists of 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. The deprecated APIs are not included in Table 1. For detailed information about each LDAP API and the list of deprecated APIs, see Chapter 2, “LDAP routines” on page 19.

| Synchronous versions of the APIs have a suffix of **_s** (for example, **ldap_add_s**). The descriptions of the
| APIs only show the asynchronous name of the API, but in general, it applies to both the synchronous and
| asynchronous versions.

*Table 1. LDAP API functions*

| **Category** | **Function name** |
|---|---|
| Initialization / termination | **ldap_init**<br>**ldap_ssl_init**, **ldap_ssl_client_init**,<br>**ldap_unbind**, **ldap_unbind_s** |
| Primitive operations | **ldap_abandon**<br>**ldap_add**, **ldap_add_s**<br>**ldap_add_ext**, **ldap_add_ext_s**<br>**ldap_compare**, **ldap_compare_s**<br>**ldap_compare_ext**, **ldap_compare_ext_s**<br>**ldap_delete**, **ldap_delete_s**<br>**ldap_delete_ext**, **ldap_delete_ext_s**<br>**ldap_extended_operation**, **ldap_extended_operation_s**<br>**ldap_modify**, **ldap_modify_s**<br>**ldap_modify_ext**, **ldap_modify_ext_s**<br>**ldap_rename**, **ldap_rename_s**<br>**ldap_result**<br>**ldap_sasl_bind**, **ldap_sasl_bind_s**<br>**ldap_search**, **ldap_search_s**, **ldap_search_st**<br>**ldap_search_ext**, **ldap_search_ext_s**<br>**ldap_simple_bind**, **ldap_simple_bind_s** |
| Error handling | **ldap_err2string**<br>**ldap_get_errno** |
| Results processing | **ldap_count_attributes**, **ldap_first_attribute**, **ldap_next_attribute**<br>**ldap_count_entries**, **ldap_first_entry**, **ldap_next_entry**<br>**ldap_count_messages**, **ldap_count_references**<br>**ldap_count_values**, **ldap_get_values**<br>**ldap_count_values_len**, **ldap_get_values_len**<br>**ldap_first_message**, **ldap_first_reference**<br>**ldap_get_dn**, **ldap_get_entry_controls_np**<br>**ldap_explode_dn**<br>**ldap_msgid**, **ldap_msgtype**<br>**ldap_next_message**, **ldap_next_reference**<br>**ldap_parse_result**, **ldap_parse_reference_np**<br>**ldap_parse_extended_result**, **ldap_parse_sasl_bind_result** |
| LDAP URL processing | **ldap_is_ldap_url**<br>**ldap_url_parse**<br>**ldap_url_search**, **ldap_url_search_s**, **ldap_url_search_st** |
| Utility functions | **ldap_control_free**, **ldap_controls_free**<br>**ldap_enetwork_domain_get**, **ldap_enetwork_domain_set**<br>**ldap_memcache_init**, **ldap_memcache_set**, **ldap_memcache_get**<br>**ldap_memcache_flush**, **ldap_memcache_update**, **ldap_memcache_destroy**<br>**ldap_memfree**, **ldap_msgfree**<br>**ldap_mods_free**, **ldap_free_urldesc**<br>**ldap_set_option**, **ldap_set_option_np**, **ldap_get_option**<br>**ldap_server_conf_save**, **ldap_server_free_list**, **ldap_server_locate**<br>**ldap_set_rebind_proc**<br>**ldap_value_free**, **ldap_value_free_len** |

Following is a description of each type of function:

**Initialization and termination functions**
| Initialization and termination functions are used to create and destroy LDAP handles.

**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 used to interpret the results that come back from an **ldap_search** operation or an **ldap_extended_operation** operation.

**LDAP URL processing functions**

The LDAP URL processing functions work with LDAP-style URLs as specified in IETF 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 client caching and setting options that determine certain runtime characteristics of the LDAP programming interface.

## Using the socksified client

The z/OS LDAP C/C++ 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** or **ldap_ssl_init** 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** or **ldap_ssl_init** LDAP APIs.

The **RESOLVER_CONFIG** environment variable specifies the shared domain name server dataset. Refer to *z/OS: Communications Server: IP Configuration Guide* 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. If both the **SOCKS_SERVER** and **SOCKS_CONF** environment variables are set,
| the **SOCKS_CONF** environment variable takes precedence.

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

*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**.
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 a z/OS dataset, the DLLs will be found in **LPALIB**.
7. | If using SSL/TLS, 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 *DSNHLQ*.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 item is also needed, and it is suggested that it be made part of the C source code:

  ```
  #pragma runopts(POSIX(ON))
  ```
- 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. Prior to initialization, SIGPIPE signals should be set to be ignored or a signal handler should be defined. TCP/IP functions can cause SIGPIPE signals. When the signal is ignored, TCP/IP reflects the signal as an EPIPE errno for the TCP/IP functions. An example of the signal ignore call looks like:

   ```
   sigignore(SIGPIPE);
   ```
2. Initialize the LDAP programming interface and the connection to the directory server that accepts the LDAP protocol using **ldap_init**.

   An example call to **ldap_init** looks like:

   ```
   LDAP *ld = ldap_init( "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.

   If **ldap_init** is called with an empty LDAP URL, it uses **ldap_server_locate** to find the appropriate server. See “ldap_server” on page 92 for more information about **ldap_server_locate**.
3. Bind to the Directory Service to establish an identity with the directory server by using **ldap_simple_bind** or **ldap_simple_bind_s**.

   An example call to **ldap_simple_bind_s** looks like:

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

   where `password` is a null-terminated character string presumably obtained from the user. The LDAP handle returned from the **ldap_init** call is used as the first parameter to the **ldap_simple_bind_s** operation. The **ldap_sasl_bind** and **ldap_sasl_bind_s** APIs are available for alternate authentication.
4. Perform LDAP operations such as add, modify, delete, compare and search using:
   - **ldap_add** and **ldap_add_s**
   - **ldap_modify** and **ldap_modify_s**
   - **ldap_delete** and **ldap_delete_s**
   - **ldap_compare** and **ldap_compare_s**
   - **ldap_search** 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 the following APIs can be used:
   - **ldap_add_ext** and **ldap_add_ext_s**
   - **ldap_delete_ext** and **ldap_delete_ext_s**
   - **ldap_compare_ext** and **ldap_compare_ext_s**
   - **ldap_search_ext** and **ldap_search_ext_s**

Examples of calls to perform LDAP operations are provided in "Performing an operation". See "Getting results" on page 13 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.

5. 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_init** is no longer valid and must not be used.

   An example of **ldap_unbind_s** looks like:

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

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

It is acceptable to perform more than one **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 19 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 wire protocol 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_REPLACE**, 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 and relocating the entry

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

Here, the RDN of the entry is changed and the entry is relocated. In this example:

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

  is the DN of the entry to be renamed.
- `"cn=Jonathan Doe",`

  is the new value of the RDN for the renamed entry.
- `"ou=Sales, o=Acme International, c=US",`

  is the DN of the new superior (parent) node under which the entry will be moved; if no relocation is being performed, this parameter should be NULL.
- `1,`

  is used to make this specification. In the example, the old RDN value is deleted.
- `NULL,`

  represent server controls.
- `NULL );`

  represent client controls.

When no controls are present, each respective parameter should be set to NULL. 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.

## 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 objectClass attribute values for all entries directly below a given entry

```
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 );
        if ( values != NULL ) {
           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_parse_result

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_parse_result**, passing in the LDAPMessage that was returned from the LDAP operation.

There is a subtle difference between using **ldap_get_errno** and **ldap_parse_result** 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_parse_result** 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 using **ldap_get_errno**. If an LDAP operation completes on a separate thread before **ldap_get_errno** examines the error code value on the current thread, the error text returned by **ldap_get_errno** will reflect the result of the LDAP operation on the other thread. Use the **ldap_parse_result** and **ldap_err2string** calls 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_parse_result

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

   /* process the error */

}
```

## Using ldap_err2string and ldap_get_option

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_get_option**. When specified with the **LDAP_OPT_ERROR_NUMBER** and **LDAP_OPT_ERROR_STRING** values, this function obtains the LDAP error code and error message. These can then be issued in a message containing the text returned by **ldap_err2string** on the standard error stream.

Be careful in a multi-threaded environment when using **ldap_get_option**. If an LDAP operation completes on a separate thread before **ldap_get_option** examines the error code or error message values on the current thread, the values returned by **ldap_get_option** will reflect the result of the LDAP operation on the other thread. Use the **ldap_parse_result** and **ldap_err2string** calls in these cases.

### Example: obtaining the 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 and character string 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 ) {
   fprintf( stderr, "Error on ldap_delete_s(): %s\n", ldap_err2string(rc) );
   ldap_get_option( ld, LDAP_OPT_ERROR_STRING, (void *)&errmsg );
   fprintf( stderr, "additional info: %s\n", errmsg );
   ldap_memfree( errmsg );
}
```

## 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** or
| **LDAP_OPT_DEBUG_STRING**. 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 );
```

or

```
rc = ldap_set_option( ld, LDAP_OPT_DEBUG_STRING, "ANY" );
```

| The value specified for **LDAP_OPT_DEBUG_STRING** is a string which can have the same values as the
| **LDAP_DEBUG** environment variable.

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

| Consult Table 2 on page 16 for a specification of the trace classes.

The second method for enabling tracing is to set the **LDAP_DEBUG** environment variable. The value for **LDAP_DEBUG** is a mask that may be specified as follows:

- A decimal value (for example, 32)
- A hexadecimal value (for example, x20 or X20)

- A keyword (for example, FILTER)
- A construct of those values using plus and minus signs to indicate inclusion or exclusion of a value. For example:
  – ’32768+8’ is the same as specifying ’32776’, or ’x8000+x8’, or ’ERROR+CONNS’
  – ’2146959359’ is the same as specifying ’ANY-STRBUF’

Table 2 lists the debug levels and the related decimal, hexadecimal, and keyword values.

*Table 2. Debug levels*

| Keyword | Decimal | Hexadecimal | Description |
|---|---:|---:|---|
| OFF | 0 | 0x00000000 | No debugging |
| TRACe | 1 | 0x00000001 | Entry and exit from routines |
| PACKets | 2 | 0x00000002 | Packet activity |
| ARGS | 4 | 0x00000004 | Data arguments from requests |
| CONNs | 8 | 0x00000008 | Connection activity |
| BER | 16 | 0x00000010 | Encoding and decoding of data, including ASCII and EBCDIC translations, if applicable |
| FILTer | 32 | 0x00000020 | Search filters |
| MESSage | 64 | 0x00000040 | Messaging subsystem activities and events |
| ACL | 128 | 0x00000080 | Access Control List activities |
| STATs | 256 | 0x00000100 | Operational statistics |
| THREad | 512 | 0x00000200 | Threading activities |
| REPLication | 1024 | 0x00000400 | Replication activities |
| PARSe | 2048 | 0x00000800 | Parsing activities |
| PERFormance | 4096 | 0x00001000 | Relational backend performance statistics |
|  | 8192 | 0x00002000 | Reserved |
| REFErral | 16384 | 0x00004000 | Referral activities |
| ERROr | 32768 | 0x00008000 | Error conditions |
| SYSPlex | 65536 | 0x00010000 | Sysplex/WLM activities |
| MULTIServer | 131072 | 0x00020000 | Multi-server activities |
| LDAPBE | 262144 | 0x00040000 | Connection between a frontend and a backend |
| STRBuf | 524288 | 0x00080000 | UTF-8 support activities |
| TDBM | 1048576 | 0x00100000 | Relational backend activities (TDBM) |
| SCHEma | 2097152 | 0x00200000 | Schema support activities (TDBM) |
| BECApabilities | 4194304 | 0x00400000 | Backend capabilities |
| CACHe | 8388608 | 0x00800000 | Cache activities |
| ANY | 2147483647 | 0x7FFFFFFF | All levels of debug |
| **Note:** The minimum abbreviation for each keyword is shown in uppercase letters. | | | |

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=ANY
```

on the z/OS shell command line prior to running the client program.

## 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.

## Client-side search results caching

Client-side search results caching is supported. It can be enabled for specific LDAP connections using the **ldap_memcache** APIs, or globally for all connections by setting environment variables. See "ldap_memcache" on page 70 for details.

## 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* for more information.

## LDAP client for Java

An industry-standard Java programming language interface exists to access 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 documentation from Sun Microsystems.

If you are just beginning to use JNDI in your applications, we recommend you use Sun's implementation located at:

`http://www.javasoft.com/products/jndi/docs.html`

| For customers who have been using the IBM JNDI LDAP service provider rather than the Sun JNDI
| service provider, migration may be necessary. The IBM JNDI service provider is no longer shipped. Use
| the Sun JNDI service provider. If you have been using the IBM JNDI service provider in order to be able to
| use SSL capabilities on z/OS, migration will be necessary. All Java SSL capabilities are provided using
| JSSE. See the JSSE Web site (`http://www.javasoft.com/products/jsse/`) for more information.

# 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 141 describes and shows the contents of the header files.
- Appendix C, "Example programs" on page 161 shows sample programs that use the LDAP APIs.
- *z/OS: Security Server LDAP Server Administration and Use* contains information about the LDAP server.

**Deprecated LDAP APIs:** Although the following APIs are still supported, their use is deprecated. Use of the newer replacement APIs is strongly encouraged:
- **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_open** (use **ldap_init**)
- **ldap_perror** (use **ldap_parse_result** or **ldap_get_option**)
- **ldap_result2error** (use **ldap_parse_result**)
- **ldap_ssl_start** (use **ldap_ssl_client_init** and **ldap_ssl_init**)

Following is a summary of the LDAP routines:

**ldap_abandon**
: Abandons an asynchronous LDAP operation that is in progress. (See "ldap_abandon" on page 28.)

**ldap_abandon_ext**
: Abandons an asynchronous operation with controls. (See "ldap_abandon" on page 28.)

**ldap_add**
: Performs an asynchronous LDAP add operation. (See "ldap_add" on page 30.)

**ldap_add_ext**
: Performs an asynchronous LDAP add operation with controls. (See "ldap_add" on page 30.)

**ldap_add_ext_s**
: Performs a synchronous LDAP add operation with controls. (See "ldap_add" on page 30.)

**ldap_add_s**
: Performs a synchronous LDAP add operation. (See "ldap_add" on page 30.)

**ldap_compare**
: Performs an asynchronous LDAP compare operation. (See "ldap_compare" on page 37.)

**ldap_compare_ext**
: Performs an asynchronous LDAP compare operation with controls. (See "ldap_compare" on page 37.)

**ldap_compare_ext_s**
: Performs a synchronous LDAP compare operation with controls. (See "ldap_compare" on page 37.)

**ldap_compare_s**
: Performs a synchronous LDAP compare operation. (See "ldap_compare" on page 37.)

**ldap_control_free**
: Frees a single LDAPControl structure. (See "ldap_memfree" on page 73.)

**ldap_controls_free**
Frees an array of LDAPControl structures. (See “ldap_memfree” on page 73.)

**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 50.)

**ldap_count_entries**
Retrieves a count of the entries in a chain of search results. (See “ldap_get_dn” on page 53.)

**ldap_count_messages**
Returns the number of messages in a result chain, as returned by **ldap_result**. (See “ldap_message” on page 74.)

**ldap_count_references**
Returns the number of continuation references in a chain of search results. (See “ldap_first_entry/reference” on page 50.)

**ldap_count_values**
Counts the number of values in an array of attribute values. (See “ldap_get_values” on page 54.)

**ldap_count_values_len**
Counts the number of pointers to values in an array of attribute values. (See “ldap_get_values” on page 54.)

**ldap_delete**
Performs an asynchronous LDAP delete operation. (See “ldap_delete” on page 39.)

**ldap_delete_ext**
Performs an asynchronous LDAP delete operation with controls. (See “ldap_delete” on page 39.)

**ldap_delete_ext_s**
Performs a synchronous LDAP delete operation with controls. (See “ldap_delete” on page 39.)

**ldap_delete_s**
Performs a synchronous LDAP delete operation. (See “ldap_delete” on page 39.)

**ldap_enetwork_domain_get**
Retrieves the user’s default eNetwork domain name. (See “ldap_enetwork_domain” on page 41.)

**ldap_enetwork_domain_set**
Sets the user’s default eNetwork domain name. (See “ldap_enetwork_domain” on page 41.)

**ldap_err2string**
Provides a textual description of an error message. (See “ldap_error” on page 42.)

**ldap_explode_dn**
Parses LDAP distinguished names. (See “ldap_get_dn” on page 53.)

**ldap_extended_operation**
Initiates an asynchronous extended operation. (See “ldap_extended_operation” on page 46.)

**ldap_extended_operation_s**
Initiates a synchronous extended operation. (See “ldap_extended_operation” on page 46.)

**ldap_first_attribute**
Begins stepping through an LDAP entry’s attributes. (See “ldap_first_entry/reference” on page 50.)

**ldap_first_entry**
Retrieves the first entry in a chain of search results. (See “ldap_get_dn” on page 53.)

**ldap_first_message**
Retrieves the first message in a result chain, as returned by **ldap_result**. (See “ldap_message” on page 74.)

**ldap_first_reference**
Retrieves the first continuation reference in a chain of search results. (See "ldap_get_dn" on page 53.)

**ldap_free_urldesc**
Deallocates an LDAP URL description obtained from a call to **ldap_url_parse**. (See "ldap_url" on page 112.)

**ldap_get_dn**
Obtains LDAP distinguished names from an LDAP entry. (See "ldap_get_dn" on page 53.)

**ldap_get_entry_controls_np**
Extracts server controls from an entry. (See "ldap_get_dn" on page 53.)

**ldap_get_errno**
Retrieves the last error code set by an LDAP operation. (See "ldap_error" on page 42.)

**ldap_get_option**
Retrieves the current value of an LDAP option. (See "ldap_init" on page 57.)

**ldap_get_values**
Retrieves attribute values from an LDAP entry in NULL-terminated character strings. (See "ldap_get_values" on page 54.)

**ldap_get_values_len**
Retrieves attribute values from an LDAP entry in pointer/length format. (See "ldap_get_values" on page 54.)

**ldap_init**
Initializes an LDAP context. (See "ldap_init" on page 57.)

**ldap_is_ldap_url**
Checks whether a character string represents an LDAP Uniform Resource Locator (URL). (See "ldap_url" on page 112.)

**ldap_memcache_destroy**
Frees all resources associated with a cache handle. (See "ldap_memcache" on page 70.)

**ldap_memcache_flush**
Removes specific cached search requests based on *base* and *scope*. (See "ldap_memcache" on page 70.)

**ldap_memcache_get**
Obtains the cache handle associated with an LDAP handle. (See "ldap_memcache" on page 70.)

**ldap_memcache_init**
Creates a client-side cache for caching LDAP search requests. (See "ldap_memcache" on page 70.)

**ldap_memcache_set**
Activates search request caching over a specific LDAP handle. (See "ldap_memcache" on page 70.)

**ldap_memcache_update**
Removes all cached search requests whose TTL has expired. (See "ldap_memcache" on page 70.)

**ldap_memfree**
Deallocates character strings allocated by the LDAP programming interface. (See "ldap_memfree" on page 73.)

**ldap_modify**
Performs an asynchronous LDAP modify operation. (See "ldap_modify" on page 76.)

**ldap_modify_ext**
: Performs an asynchronous LDAP modify operation with controls. (See "ldap_modify" on page 76.)

**ldap_modify_ext_s**
: Performs a synchronous LDAP modify operation with controls. (See "ldap_modify" on page 76.)

**ldap_modify_s**
: Modifies LDAP entries synchronously. (See "ldap_modify" on page 76.)

**ldap_mods_free**
: Deallocates a NULL-terminated array of modification structures. (See "ldap_modify" on page 76.)

**ldap_msgfree**
: Deallocates the memory allocated for a result. (See "ldap_rename" on page 82.)

**ldap_msgid**
: Retrieves the message ID associated with an LDAP message. (See "ldap_result" on page 85.)

**ldap_msgtype**
: Retrieves the next attribute type name in an LDAP result. (See "ldap_result" on page 85.)

**ldap_next_attribute**
: Deallocates a NULL-terminated array of modification structures. (See "ldap_first_entry/reference" on page 50.)

**ldap_next_entry**
: Retrieves the next entry in a chain of search results to parse. (See "ldap_get_dn" on page 53.)

**ldap_next_message**
: Retrieves the next message in a result chain, as returned by **ldap_result**. (See "ldap_message" on page 74.)

**ldap_next_reference**
: Retrieves the next continuation reference in a chain of search results. (See "ldap_get_dn" on page 53.)

**ldap_parse_extended_result**
: Extracts information from extended operation results. (See "ldap_parse_result" on page 79.)

**ldap_parse_reference_np**
: Extracts information from a continuation reference. (See "ldap_get_dn" on page 53.)

**ldap_parse_result**
: Extracts information from results. (See "ldap_parse_result" on page 79.)

**ldap_parse_sasl_bind_result**
: Extracts server credentials from SASL bind results. (See "ldap_parse_result" on page 79.)

**ldap_rename**
: Performs an asynchronous LDAP rename operation. (See "ldap_rename" on page 82.)

**ldap_rename_s**
: Performs a synchronous LDAP rename operation. (See "ldap_rename" on page 82.)

**ldap_result**
: Waits for the result of an LDAP operation. (See "ldap_result" on page 85.)

**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 32.)

**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 32.)

**ldap_search**
: Performs an asynchronous LDAP search operation. (See “ldap_search” on page 87.)

**ldap_search_ext**
: Performs an asynchronous LDAP search operation with controls. (See “ldap_search” on page 87.)

**ldap_search_ext_s**
: Performs a synchronous LDAP search operation with controls. (See “ldap_search” on page 87.)

**ldap_search_s**
: Performs a synchronous LDAP search operation. (See “ldap_search” on page 87.)

**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 87.)

**ldap_server_conf_save**
: Stores published LDAP server information into the local configuration file. (See “ldap_server” on page 92.)

**ldap_server_free_list**
: Frees the list of servers and associated LDAPServerInfo structures returned from **ldap_server_locate**. (See “ldap_server” on page 92.)

**ldap_server_locate**
: Returns a published list of candidate LDAP servers. (See “ldap_server” on page 92.)

**ldap_set_option**
: Sets the value of an LDAP option. (See “ldap_init” on page 57.)

**ldap_set_option_np**
: Sets the value of an LDAP option. This API is nonportable. (See “ldap_init” on page 57.)

**ldap_set_rebind_proc**
: Establishes a call-back function for rebinding during referrals chasing. (See “ldap_bind” on page 32.)

**ldap_simple_bind**
: Binds to an LDAP server asynchronously using simple authentication in order to perform directory operations. (See “ldap_bind” on page 32.)

**ldap_simple_bind_s**
: Binds to an LDAP server synchronously using simple authentication in order to perform directory operations. (See “ldap_bind” on page 32.)

**ldap_ssl_client_init**
: Initializes the SSL library. (See “ldap_ssl” on page 107.)

**ldap_ssl_init**
: Initializes an SSL connection. (See “ldap_ssl” on page 107.)

**ldap_unbind**
: Unbinds from an LDAP server asynchronously and deallocates an LDAP handle. (See “ldap_init” on page 57.)

**ldap_unbind_s**
: Unbinds from an LDAP server synchronously and deallocates an LDAP handle. (See “ldap_init” on page 57.)

**ldap_url_parse**
: Breaks down an LDAP URL into its component pieces. (See “ldap_url” on page 112.)

**ldap_url_search**
: Initiates an asynchronous LDAP search based on an LDAP URL. (See “ldap_url” on page 112.)

**ldap_url_search_s**
Initiates a synchronous LDAP search based on an LDAP URL. (See “ldap_url” on page 112.)

**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 112.)

**ldap_value_free**
Deallocates values returned by **ldap_get_values**. (See “ldap_get_values” on page 54.)

**ldap_value_free_len**
Deallocates values returned by **ldap_get_values_len**. (See “ldap_get_values” on page 54.)

## 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. 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, *PLDAPControl;
```

The LDAPControl fields have the following definitions:

*ldctl_oid*
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**.

## Supported client controls

Currently, the only client controls supported by this library are: **ibm-serverHandledSearchRequest**, | **ibm-saslBindDigestUserName** or **ibm-saslBindCramUserName**, and **ibm-saslBindDigestRealmName** | or **ibm-saslBindCramRealmName**.

**Name: ibm-serverHandledSearchRequest**

**Numeric OID:** 1.3.18.0.2.10.7

**Purpose:** Provides the ability to selectively bypass cache usage per search request.

**Criticality: TRUE** or **FALSE**. If **TRUE**, then if used on an operation which does not support this control, the request fails with **LDAP_UNAVAILABLE_CRITICAL_EXTENSION**. If **FALSE**, operations which do not | support this control will ignore its presence and still service the request. This control is only supported by | LDAP search operations. (See “ldap_search” on page 87.)

**Value:**

```
ibm-serverHandledSearchRequest ::= SEQUENCE {
     cacheResults   BOOLEAN DEFAULT FALSE
}
```

**Meaning:**

- If the control is not present, the search request can be handled from the cache, if it is cached. If the search request is not cached, the search is passed on to the server, and the results can be cached.
- If the control is present, then if the *cacheResults* flag is **FALSE** (or not present, that is, an empty SEQUENCE), then the client must bypass the cache, send the request to the server, and bypass adding the results to the cache.
- If the control is present, then if the *cacheResults* flag is **TRUE**, then whether or not the search request is cached, the search is passed onto the server, and the results can be cached.

**Notes:**

1. The *cacheResults* must be a BER encoded sequence, if specified. For coding convenience, **ldap.h** defines the constants **BER_ENCODED_BOOLEAN_TRUE** and **BER_ENCODED_BOOLEAN_FALSE**. Additionally, the **IBM_SERVER_HANDLED_SEARCH_REQUEST_OID** constant is defined and represents the numeric OID for this control. Following is an example of defining an **ibm-serverHandledSearchRequest** control:

   ```
   static LDAPControl skipCacheControl = { IBM_SERVER_HANDLED_SEARCH_REQUEST_OID, /* OID */
        {sizeof(BER_ENCODED_BOOLEAN_FALSE)-1, BER_ENCODED_BOOLEAN_FALSE},         /* false */
        LDAP_OPT_ON                                                               /* critical */
   };
   ```

   | When **skipCacheControl** is supplied as a client control on a search request, the search request will | not be satisfied from a client cache. Similarly, the search results will not be stored in a client cache.
2. This control is only supported by the LDAP search operations. (See “ldap_search” on page 87.)
3. This control is only applicable if client-side caching is enabled. (See “ldap_memcache” on page 70.)

| **Name: ibm-saslBindDigestUserName** or **ibm-saslBindCramUserName**

| **Numeric OID:** 1.3.18.0.2.10.13

| **Purpose:** Provides the ability to specify the user name authentication identity for a CRAM-MD5 or
| DIGEST-MD5 SASL authentication bind.

| **Criticality: TRUE** or **FALSE**. If **TRUE**, then if used on an operation which does not support this control,
| the request fails with **LDAP_UNAVAILABLE_CRITICAL_EXTENSION**. If **FALSE**, operations which do not
| support this control will ignore its presence and still service the request. This control is only supported by
| LDAP bind operations. (See "ldap_bind" on page 32.)

**Value:**

```
| ibm-saslBindDigestUserName {
|      userName LDAPString
| }
| ibm-saslCramDigestUserName {
|      userName LDAPString
| }
```

| The string *userName* is a NULL terminated string in either local code page or UTF-8 depending upon the
| setting in the client.

**Meaning:**

If the control is present and CRAM-MD5 or DIGEST-MD5 authentication is desired, then the *userName* is the identity used for authentication binding.

**Notes:**

1. For coding convenience, **ldap.h** defines the constant **IBM_CLIENT_MD5_USER_NAME_OID** as the numeric OID for this control. Following is an example of defining an **ibm-saslBindDigestUserName** or **ibm-saslBindCramUserName** control:

   ```
   static LDAPControl userControl = { IBM_CLIENT_MD5_USER_NAME_OID, /* OID */
        { strlen("jon"), "jon" },                                /* username */
        LDAP_OPT_OFF                                             /* non-critical */
   };
   ```

   When **userControl** is supplied as a client control on a bind request, the bind request will use the username authentication identity for performing the DIGEST-MD5 or CRAM-MD5 SASL bind that is desired.
2. This control is only supported by the LDAP bind operation. (See "ldap_bind" on page 32.)

| **Name: ibm-saslBindDigestRealmName** or **ibm-saslBindCramRealmName**

| **Numeric OID:** 1.3.18.0.2.10.12

| **Purpose:** Provides the ability to specify the realm name for a CRAM-MD5 or DIGEST-MD5 SASL
| authentication bind.

| **Criticality: TRUE** or **FALSE**. If **TRUE**, then if used on an operation which does not support this control,
| the request fails with **LDAP_UNAVAILABLE_CRITICAL_EXTENSION**. If **FALSE**, operations which do not
| support this control will ignore its presence and still service the request. This control is only supported by
| LDAP bind operations. (See "ldap_bind" on page 32.)

**Value:**

```
ibm-saslBindDigestRealmName ::=SEQUENCE {
     realmName LDAPString
}
ibm-saslCramRealmName ::=SEQUENCE {
     realmName LDAPString
}
```

**Meaning:**

If the control is present and CRAM-MD5 or DIGEST-MD5 authentication is desired, then the *realmName* is used to select a realm in which to bind.

**Notes:**

1. For coding convenience, **ldap.h** defines the constant **IBM_CLIENT_MD5_REALM_NAME_OID** as the numeric OID for this control. Following is an example of defining an **ibm-saslBindDigestRealmName** or **ibm-saslBindCramRealmName** control:

   ```
   static LDAPControl realmControl = { IBM_CLIENT_MD5_REALM_NAME_OID, /* OID */
        { strlen("myrealm.ibm.com"), "myrealm.ibm.com" },           /* realm name */
        LDAP_OPT_OFF                                                 /* non-critical */
   };
   ```

   When **realmControl** is supplied as a client control on a bind request, the bind request will use the realm name for performing the DIGEST-MD5 or CRAM-MD5 bind that is desired.
2. This control is only supported by the LDAP bind operation. (See "ldap_bind" on page 32.)

## Using RACF® data

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

## 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_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 24 for more information about server controls.

*clientctrls*
Specifies a list of LDAP client controls. This parameter may be set to NULL. See “Supported client controls” on page 25 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 42 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_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 76.

*serverctrls*
Specifies a list of LDAP server controls. This parameter may be set to NULL. See "LDAP controls" on page 24 for more information about server controls.

*clientctrls*
Specifies a list of LDAP client controls. This parameter may be set to NULL. See "Supported client controls" on page 25 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 42 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 42 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_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)

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_ssl_init** or **ldap_init**.

*who*
Specifies the distinguished name of the entry as which to bind. If the mechanism chosen is either **LDAP_MECHANISM_CRAM** or **LDAP_MECHANISM_DIGEST**, then this represents the authorization identity distinguished name which will be used to perform the authentication checks on the LDAP server.

*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. If the mechanism is either **LDAP_MECHANISM_CRAM** or **LDAP_MECHANISM_DIGEST**, then the *bv_val* field of the berval structure should point to the credentials that will be used for the bind of the *userName* specified in the **LDAPControl ibm-saslBindDigestUserName** or **ibm-saslCramDigestUserName**.

*mechanism*
Although a variety of mechanisms have been IANA (Internet Assigned Numbers Authority) registered, the only mechanisms supported at this time are: **LDAP_SASL_SIMPLE**, **LDAP_MECHANISM_EXTERNAL**, **LDAP_MECHANISM_GSSAPI**, **LDAP_MECHANISM_CRAM**, and **LDAP_MECHANISM_DIGEST**.

In order to use the UTF-8 versions of these mechanism strings, it is necessary to set the **LDAP_OPT_UTF8_IO** option to **LDAP_OPT_ON** with the **ldap_set_option** API.

The following table shows the strings that are defined in the **ldap.h** header file that should be used depending upon the bind mechanism that is desired.

| Mechanism | LDAP_OPT_OFF | LDAP_OPT_ON |
|---|---|---|
| SIMPLE | LDAP_SASL_SIMPLE | LDAP_SASL_SIMPLE |
| EXTERNAL | LDAP_MECHANISM_EXTERNAL | LDAP_MECHANISM_EXTERNAL_UTF8 |
| GSSAPI | LDAP_MECHANISM_GSSAPI | LDAP_MECHANISM_GSSAPI_UTF8 |
| CRAM-MD5 | LDAP_MECHANISM_CRAM | LDAP_MECHANISM_CRAM_UTF8 |
| DIGEST-MD5 | LDAP_MECHANISM_DIGEST | LDAP_MECHANISM_DIGEST_UTF8 |

*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 24 for more information about server controls.

*clientctrls*
Specifies a list of LDAP client controls. This parameter may be set to NULL. See "Supported client controls" on page 25 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**, 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 42). 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, certificate, CRAM-MD5 and DIGEST-MD5 authentication over LDAP through the use of the Simple Authentication Security Layer (SASL). The **ldap_sasl_bind_s** API can also be used for Kerberos Version 5 binds. For information on setting up Kerberos on your z/OS system, see *z/OS: Security Server Network Authentication Service Administration*.

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. By setting *mechanism* to **LDAP_MECHANISM_GSSAPI**, the SASL bind
| request is interpreted as a request for Kerberos Version 5 authentication. By setting *mechanism* to
| **LDAP_MECHANISM_CRAM**, the SASL bind request will be interpreted as a request for CRAM-MD5
| authentication. By setting *mechanism* to **LDAP_MECHANISM_DIGEST**, the SASL bind request will be
| interpreted as a request for DIGEST-MD5 authentication.

With this implementation, there are several primary reasons for using the SASL bind APIs. The first reason is to use the client authentication mechanism provided by SSL to strongly authenticate to the directory server, using the client's X.509 certificate. 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). 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`...)

The second reason is to use the authentication mechanism provided by Kerberos Version 5. This method requires the application to have obtained a valid Ticket Granting Ticket (TGT) prior to invoking the **ldap_sasl_bind_s** API. Once a client and server that supports this mechanism have authenticated each other, the client can access the directory with the identity contained in the credentials used in the authentication process. To see how this mapping takes place in the directory, see *z/OS: Security Server LDAP Server Administration and Use*.

| The third reason is to use the authentication mechanisms provided by either CRAM-MD5 or DIGEST-MD5.
| Both of these methods provide different manners of hashing the password with the challenge that is sent
| back from the server instead of sending the password in the clear.

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**).

## 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_simple_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. If a referral is sent from the server to the client on a CRAM-MD5 or DIGEST-MD5 bind,
| the referral will not be followed. CRAM-MD5 and DIGEST-MD5 binds do not follow referrals.

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 methods for rebinding are **LDAP_AUTH_SIMPLE** and **LDAP_AUTH_SASL_30**. **LDAP_AUTH_SASL_30** can only be used if the desired rebind mechanism is Kerberos Version 5. In this scenario, the *dnp* parameter and the *passwdp* parameter should be set to **NULL** in the function. The credentials that will be used for the automatic rebind will be the credentials that were used on the current bind. Also, the client can only rebind using Kerberos if the current bind context for the client was Kerberos (that is, you cannot use **LDAP_AUTH_SASL_30**, Kerberos Version 5, for the rebind method if you are currently bound to the directory using a DN and password, SSL, or anonymous). Also, the client cannot remove the credentials it bound with from its credential cache if it wants to rebind using Kerberos. The credentials must remain for the life of the current bind.

**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 42 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 42 for possible values.

## Related topics

**ldap_error control**

## 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_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 54), 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 24 for more information about server controls.

*clientctrls*
Specifies a list of LDAP client controls. This parameter may be set to NULL. See "Supported client controls" on page 25 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 42 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 42 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_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 24 for more information about server controls.

*clientctrls*
Specifies a list of LDAP client controls. This parameter may be set to NULL. See “Supported client controls” on page 25 for more information about client controls.

#### Output

*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 42 for possible values.

The **ldap_delete_s** API returns **LDAP_SUCCESS** if successful, otherwise an error code is returned. See “ldap_error” on page 42 for possible values.

## Related topics

**ldap_error**

## ldap_enetwork_domain

**ldap_enetwork_domain_set**
**ldap_enetwork_domain_get**

### Purpose

Manage a user's eNetwork domain.

### Format

```
#include <ldap.h>

int ldap_enetwork_domain_set(
        char *edomain,
        char *filename)

int ldap_enetwork_domain_get(
        char **edomainp,
        char *filename)
```

### Parameters

#### Input

*edomain*
Specifies the eNetwork domain.

*filename*
Specifies the fully-qualified file name where to store or retrieve a user's eNetwork domain. If NULL, the default file (`/home/`*`user`*`/ldap_user_info`) is used.

| #### Output

| *edomainp*
| Specifies the eNetwork domain, as returned from **ldap_enetwork_domain_get**.

### Usage

The **ldap_enetwork_domain_set** API is used to set the user's default eNetwork domain name. The **ldap_enetwork_domain_get** API is used to retrieve the user's default eNetwork domain name. If an error occurs, no string is returned. To free the returned string, use **ldap_memfree**.

The eNetwork domain name (along with the user's default Domain Name Service (DNS) domain name) is used to identify the user's LDAP authentication domain. For example, if a user's eNetwork domain name is `chicago`, and the user's DNS domain is `midwest.illinois.com`, then information can be published in DNS that associates `ldap.chicago.midwest.illinois.com` with a collection of LDAP servers (one or more masters and replicas). This permits applications to easily find an appropriate LDAP authentication server by using the **ldap_server_locate** API.

An application can retrieve the eNetwork domain name by calling **ldap_enetwork_domain_get**.

### Error conditions

The **ldap_enetwork_domain_set** and **ldap_enetwork_domain_get** APIs return **LDAP_SUCCESS** if successful, otherwise an error code is returned. See "ldap_error" on page 42 for possible values.

### Related topics

**ldap_server_locate**
**ldap_error**
**ldap_memfree**

## 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_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 87).

*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 3. 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 | A 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 3. 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 | Credentials that were not valid 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 3. 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 | A filter that was not valid 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 | A LDAP routine was called with a bad parameter (for example, a NULL *ld* pointer). |
| **LDAP_NO_MEMORY** | Out of memory | A memory allocation call (for example, malloc) 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 which is 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** API was successfully called previously in this process. |
| **LDAP_SSL_INITIALIZE_FAILED** | SSL 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 | A SSL parameter that was not valid was previously specified. |
| **LDAP_SSL_HANDSHAKE_FAILED** | SSL handshake with the server failed | Failed to connect to SSL server. |
| **LDAP_SSL_GET_CIPHER_FAILED** | Failed to retrieve cipher code | Failed to retrieve the cipher code for SSL. |
| **LDAP_NO_EXPLICIT_OWNER** | No explicit owner found | An explicit owner does not exist. |
| **LDAP_NO_EXPLICIT_ACL** | No explicit acl found | An explicit ACL does not exist. |
| **LDAP_SSL_NOT_AVAILABLE**, | SSL support is not available | SSL support is not available. |

## Related topics

**ldap_memfree**
**ldap_parse_result**

## ldap_extended_operation

**ldap_extended_operation**
**ldap_extended_operation_s**

### Purpose

Perform extended operations.

### Format

```
#include <ldap.h>

int ldap_extended_operation(
        LDAP *ld,
        const char *reqoid,
        const struct berval *reqdata,
        LDAPControl **serverctrls,
        LDAPControl **clientctrls,
        int *msgidp)

int ldap_extended_operation_s(
        LDAP *ld,
        const char *reqoid,
        const struct berval *reqdata,
        LDAPControl **serverctrls,
        LDAPControl **clientctrls,
        char **resultoidp,
        struct berval **resultdatap)
```

### Parameters

#### Input

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

*reqoid*
Specifies the dotted-OID text string that identifies the extended operation to be performed by the server.

*reqdata*
Specifies the arbitrary data required by the extended operation. (If NULL, no data is sent to the server.)

*serverctrls*
Specifies a list of LDAP server controls. This parameter may be set to NULL. See “LDAP controls” on page 24 for more information about server controls.

*clientctrls*
Specifies a list of LDAP client controls. This parameter should be set to NULL because client controls are not currently supported for extended operations. See “Supported client controls” on page 25 for more information about client controls.

#### Output

*msgidp*
Specifies a pointer to a result parameter that is set to the message ID of the request if the **ldap_extended_operation** API is successfully sent to the server. Use **ldap_result** and then **ldap_parse_extended_result** to get the results.

*resultoidp*
Specifies a result parameter is set to point to a character string that is set to an allocated, dotted-OID text string returned from the server. Free this string using the **ldap_memfree** API. If no OID is returned, *resultoidp* is set to NULL.

*resultdatap*
Specifies a pointer to a result parameter that is set to a pointer to a berval structure pointer that is set to an allocated copy of the data returned by the server. Free this struct berval using the code supplied in the **Notes** below. If no data is returned, *resultdatap* is set to NULL.

## Usage

The **ldap_extended_operation** API is used to initiate an asynchronous extended operation, which returns **LDAP_SUCCESS** if the extended operation was successfully sent, or an LDAP error code if not. If successful, the **ldap_extended_operation** API places the message ID of the request in *msgidp*. A subsequent call to **ldap_result** can be used to obtain the result of the extended operation, which can then be passed to **ldap_parse_extended_result** to obtain the OID and data contained in the response.

The **ldap_extended_operation_s** function is used to initiate a synchronous extended operation, which returns the result of the operation, either **LDAP_SUCCESS** if the operation was successful, or another LDAP error code if it was not. The *retoid* and *retdata* parameters are filled in with the OID and data from the response. If no OID or data was returned, these parameters are set to NULL, respectively.

If the LDAP server does not support the extended operation, the server will reject the request.

To determine if the requisite extended operation is supported by the server, get the root DSE of the LDAP server, and check for the **supportedExtension** attribute. If the values for this attribute include the object identifier (OID) of your extended operation, then the server supports the extended operation. If the **supportedExtension** attribute is not present in the root DSE, then the server is not configured to support any extended operations. See “Searching a server’s root DSE” on page 140 for details on accessing the root DSE.

### Notes

These routines allocate storage. Use **ldap_memfree** to free the returned OID. Use the following code to free the returned struct berval:

```
if ( resultdatap != NULL ) {
    if ( resultdatap->bv_val != NULL ) {
        ldap_memfree( resultdatap->bv_val );
    }
    ldap_memfree( (char *)resultdatap );
}
```

## Error conditions

The **ldap_extended_operation_s** API returns the LDAP error code resulting from the operation.

The **ldap_extended_operation** API returns -1 instead of a valid *msgid* if an error occurs, setting the session error in the LD structure, which can be obtained by using **ldap_get_errno**.

See “ldap_error” on page 42 for more details.

## Related topics

**ldap_error**
**ldap_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_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 42 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 42 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_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 87).

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

*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.

### Output

*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**.

*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.

## 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
         */
{
```

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 42 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 42 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 42 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_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 42 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_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.

### Output

*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 42 for possible values.

## Related topics

**ldap_first_entry/reference**
**ldap_first_attribute**
**ldap_error**

## ldap_init

**ldap_init**
**ldap_open** (deprecated)
**ldap_unbind**
**ldap_unbind_s**
**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_unbind(
        LDAP *ld)


int ldap_unbind_s(
        LDAP *ld)


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


int ldap_set_option_np(
        LDAP *ld,
        int optionToSet,
        void *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.

The *host* parameter can also be specified as a single LDAP URL. The format of the LDAP URL is:

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

where:

*host* Is an optional DNS-style host name.

*port* Is an optional port number.

*dn* Is the distinguished name.

The *attributes*, *scope*, and *filter* portions of the URL are ignored by this operation.

The port number specified in the URL overrides the **ldap_init** *port* parameter. If a port number is not specified in the URL, the default port 389 is used (636 for the SSL "ldaps" URL format).

If the URL host name is omitted, **ldap_init** and **ldap_open** attempt to locate an LDAP server to communicate with through an internal call to the **ldap_server_locate** function. In this case, the *dn* field (if specified) is used as input to **ldap_server_locate** to narrow the scope of eligible LDAP servers. See "ldap_server_locate usage by ldap_init, and ldap_ssl_init" on page 105 for details.

**Note:** Calling **ldap_init** or **ldap_open** using the SSL URL format (**ldaps**) is equivalent to calling the **ldap_ssl_init** function with NULL specified as the **ldap_ssl_init** *name* parameter. See "ldap_ssl" on page 107 for details, including SSL initialization requirements.

*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**.

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

*optionToSet*
Specifies which LDAP option's value should be set. See "Setting and getting session settings" on page 60 for the list of supported options.

*optionToGet*
Specifies which LDAP option's value should be returned. See "Setting and getting session settings" on page 60 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 4. 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 * |

*Table 4. The optionValue parameter specifications (continued)*

| *optionToSet* **or** *optionToGet* | **ldap_set_option (Version 3)** | **ldap_set_option (Version 2)** | **ldap_set_option_np** | **ldap_get_option** |
|---|---|---|---|---|
| **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_DEBUG_STRING** | char * | char * | char * | char ** |
| **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_DELEGATION** | int (ON/OFF) | N/A | int (ON/OFF) | int * |
| **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.

TCP/IP can cause a SIGPIPE signal to be generated when a peer closes their connection unexpectedly. In order for the TCP/IP function calls to be notified, the SIGPIPE signal should be ignored. This causes an error return and EPIPE errno to be returned to the TCP/IP functions instead of creating the SIGPIPE signal. The application should code the signal ignore prior to invoking the **ldap_init** API. An example of the signal ignore call looks like:

```
sigignore(SIGPIPE);
```

For SSL/TLS, the equivalent of ldap_init is **ldap_ssl_init.** The **ldap_ssl_init** API is used to initialize a secure session with a server. See “ldap_ssl” on page 107 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/TLS) 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_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.

**ldap_init**

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**.

## Environment variables affecting session settings

There are three environment variables that can affect the session settings. One, **LDAP_DEBUG**, is discussed in “Tracing” on page 15. Setting the **LDAP_DEBUG** environment variable has the same effect as calling **ldap_set_option** to set the **LDAP_OPT_DEBUG_STRING** 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 65 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 67 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 debug options as a character string (**LDAP_OPT_DEBUG_STRING**)
- 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 whether the client passes Kerberos Version 5 delegated credentials to the server. (**LDAP_OPT_DELEGATION**)
- 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**)
- 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 68 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 68.

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_ACL**
- **LDAP_DEBUG_STATS**
- **LDAP_DEBUG_PARSE**
- **LDAP_DEBUG_CACHE**

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. When setting either the **LDAP_OPT_DEBUG** or **LDAP_OPT_DEBUG_STRING** options, the *ld* parameter can be specified as NULL. 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_DEBUG_STRING

Specifies the level of debug trace for the LDAP library as a character string.

The **optionValue** parameter is the address of a character string that specifies the debug level as a mask:

- A decimal value (for example, 32)
- A hexadecimal value (for example, x20 or X20)
- A keyword (for example, FILTER)
- A construct of those values using plus and minus signs to indicate inclusion or exclusion of a value. For example:
  - ’32768+8’ is the same as specifying ’32776’, or ’x8000+x8’, or ’ERROR+CONNS’
  - ’2146959359’ is the same as specifying ’ANY-STRBUF’
  - By beginning the debug level with a minus sign, you can deactivate debug collection for the various types. -CONNS modifies an existing debug level by deactivating connection traces.
  - By beginning the debug level with a plus sign, you can activate debug collection for the various types. +CONNS modifies an existing debug level by activating connection traces.

The debug level may be set at a number of different times.

- The initial debug level is OFF.
- Prior to starting the client program, the **LDAP_DEBUG** environment variable may be set. The client API uses this value first. For example:
  ```
  export LDAP_DEBUG='ERROR+TRACE'
  ```
- When set by **ldap_set_option** or **ldap_set_option_np** using the **LDAP_OPT_DEBUG** option. The mask value specified with this option replaces the existing debug mask.
- When set by **ldap_set_option** or **ldap_set_option_np** using the **LDAP_OPT_DEBUG_STRING** option. The mask value specified with this option replaces, or incrementally adds or deletes debug levels from the current debug mask depending upon whether the mask value begins with a plus or minus sign.

**LDAP_OPT_DEBUG_STRING** is a global option (it does not pertain to any particular LDAP handle), whereas the other options pertain to a specific LDAP handle. When setting either the **LDAP_OPT_DEBUG** or **LDAP_OPT_DEBUG_STRING** options, the *ld* parameter can be specified as NULL.

If the debug level specified on the **ldap_set_option** or **ldap_set_option_np** function contains a debug level that is not valid, the debug level will not be changed and an error message will be printed to the STDERR. The debug level may be obtained as a character string. It is up to the caller of **ldap_get_option** to free the storage when its usage is complete.

Examples:

```
char * resultingDebug;
char * addConnsBer = "+Conns+16";
/* Example of adding a debug level to the existing debug level */
ldap_set_option( ld, LDAP_OPT_DEBUG_STRING, addConnsBer );
ldap_get_option( ld, LDAP_OPT_DEBUG_STRING, &resultingDebug );
printf( "New debug level: %s\n", resultingDebug );
ldap_memfree( resultingDebug);
```

### 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_DES_SHA_EX** “09”
- **LDAP_SSL_3DES_SHA_US** “0A”
- | **LDAP_SSL_RSA_AES_128_SHA** “2F”
- | **LDAP_SSL_RSA_AES_256_SHA** “35”

| **Note:** **LDAP_SSL_DES_SHA_US** has been deprecated. **LDAP_SSL_DES_SHA_EX** should be used instead.

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.

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_DELEGATION

Specifies whether the client passes Kerberos Version 5 delegated credentials to the LDAP server. Use this option if you want to allow the server to use the client's credentials for requests. Note that the server may or may not support this capability. See the server documentation for the server you are contacting. The z/OS LDAP Server does not support this capability.

The option can be set to one of the constants **LDAP_OPT_ON** or **LDAP_OPT_OFF**. By default, the option is set to **LDAP_OPT_OFF**. This option is only valid if it is set prior to calling the **ldap_sasl_bind_s** API.

Examples:

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

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

### 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:

**ldap_init**

```
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_EXT_ERROR, (void *) &exterror);
```

Returns errors reported by the SSL library.

### 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**

## Error conditions

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

The **ldap_unbind** 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 42 for possible values.

The **ldap_unbind_s** API returns **LDAP_SUCCESS** if successful, otherwise an error code is returned. See “ldap_error” on page 42 for possible values.

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.

## Related topics

**ldap_bind**
**ldap_server**

## ldap_memcache

**ldap_memcache_init**
**ldap_memcache_set**
**ldap_memcache_get**
**ldap_memcache_flush**
**ldap_memcache_update**
**ldap_memcache_destroy**

## Purpose

Support client-side caching of LDAP search results.

## Format

```
#include <ldap.h>

int ldap_memcache_init(
        unsigned long ttl,
        unsigned long size,
        char **baseDNs,
        struct ldap_thread_fns *thread_fns,
        LDAPMemCache **cachep)

int ldap_memcache_set(
        LDAP *ld,
        LDAPMemCache *cache)

int ldap_memcache_get(
        LDAP *ld,
        LDAPMemCache **cachep)

void ldap_memcache_flush(
        LDAPMemCache *cache,
        char *dn,
        int scope)

void ldap_memcache_update(
        LDAPMemCache *cache)

void ldap_memcache_destroy(
        LDAPMemCache *cache)
```

## Parameters

### Input

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

*ttl* Specifies the time limit (in seconds) that data will remain cached. Set to 0 for no time limit.

*size*
Specifies the size limit (in bytes) for the cache. Set to 0 for no size limit.

*baseDNs*
Specifies a list of search base distinguished names (DNs) for the specific search requests to be cached. (See the description in "ldap_search" on page 87.) Specify NULL for all search requests to be cached.

*dn* Specifies the base DN of a previously cached search request.

*scope*
Specifies the scope of a search request. The value can be **LDAP_SCOPE_BASE**, **LDAP_SCOPE_ONELEVEL**, or **LDAP_SCOPE_SUBTREE**.

*thread_fns*
Provided solely for prototype consistency with other SDKs. This parameter is ignored.

*cache*
Specifies a cache handle pointer.

**Output**

*cachep*
Specifies the storage location to store a cache handle pointer.

## Usage

The **ldap_memcache_init** function creates a client-side cache and associated cache handle (**LDAPMemCache**). A pointer to the cache handle is stored at the address specified by the *cachep* output parameter. To activate caching of search requests, the cache handle must be associated with one or more LDAP handles through the *cache* parameter of **ldap_memcache_set** (to deactivate caching, specify NULL for the cache parameter). Once a cache handle is assigned to one or more LDAP handles, search requests issued over the associated LDAP handle or handles will be cached. Subsequent matching search requests will be satisfied from the cache.

Below is the criteria used to determine if a search request matches a cached search request:

- Host name and port of the LDAP server being queried.
- Search request parameters (base, scope, filter, and so on). See “ldap_search” on page 87 for the complete list of search parameters.
- Certain LDAP programming interface option values (See Table 4 on page 58 for details on these settings):
  - **LDAP_OPT_REFERRALS**. If set to **LDAP_OPT_ON** (the default), the settings for **LDAP_OPT_REFHOPLIMIT** and **LDAP_OPT_REBIND_FN** settings are also compared.
  - **LDAP_OPT_PROTOCOL_VERSION**
- Bind method and identity. (See “ldap_bind” on page 32.) Note that bind credentials are not stored in the cache.

To explicitly remove expired search requests from the cache, use **ldap_memcache_update**. Note that this function is periodically performed internally by the caching support itself.

Use **ldap_memcache_flush** to remove cached search requests whose search base DN is within the scope identified by the combination of the search base DN (*dn*) and *scope* parameters. If the *dn* parameter is NULL, the *scope* parameter is ignored and the entire cache is drained.

The **ldap_memcache_destroy** API frees all resources associated with a cache handle. After this API is called, the cache handle is no longer valid.

To bypass cache usage for specific search requests, the **ibm-serverHandledSearchRequest** client control can be specified. (See “Supported client controls” on page 25.)

To disable cache usage by a specific LDAP handle, use the **ldap_memcache_set** API specifying NULL for the *cache* parameter.

**Notes:**

1. Caching should be disabled for an LDAP handle prior to calling **ldap_unbind**.
2. To trace client cache activity, enable debug level **LDAP_DEBUG_CACHE**.

**Global cache support:**

When activated, a global cache is a single client-side cache shared by all LDAP handles. A global cache is activated by setting environment variables. Therefore, an application can make use of a global cache without using the caching APIs described above. These environment variables are:

**ldap_memcache**

- **LDAP_CLIENT_CACHE**
  - Set to **ON** to enable, **OFF** to disable.
- **LDAP_CLIENT_CACHE_TTL**
  - Equivalent to the *ttl* parameter of **ldap_memcache_init**.
- **LDAP_CLIENT_CACHE_MAX_SIZE**
  - Equivalent to the size parameter of **ldap_memcache_init**.

**Global cache usage notes:**

- Both **LDAP_CLIENT_CACHE** and **LDAP_CLIENT_CACHE_TTL** must be explicitly set in order for a global cache to be enabled.
- The decision to create and activate a global cache is determined once as part of GLDCLDAP DLL initialization. After this point, changes to these environment variables have no effect.
- An LDAP handle initialized to use a global cache can be modified to use a different cache using **ldap_memcache_set**.
- All search requests issued against LDAP handles using a global cache are cached. Selective search request filtering by search base is not possible (**ldap_memcache_init** provides this option).
- Calls to **ldap_memcache_destroy** against the global cache handle causes the global cache to be drained; it does not invalidate the cache handle (the global cache is still usable).

## Error conditions

The **ldap_memcache_init**, **ldap_memcache_set** and **ldap_memcache_get** APIs return **LDAP_SUCCESS** if successful, otherwise an error code is returned. See “ldap_error” on page 42 for possible values.

## Related topics

**ldap_search**
**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_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 87).

*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 42 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

*ld* Specifies the LDAP handle returned by a previous call to **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 54.

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 24 for more information about server controls.

*clientctrls*
Specifies a list of LDAP client controls. This parameter may be set to NULL. See “Supported client controls” on page 25 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 42 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 42 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**
**ldap_parse_extended_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)


int ldap_parse_extended_result(
        LDAP *ld,
        LDAPMessage *res,
        char **resultoidp,
        struct berval **resultdatap,
        int freeit)
```

### Parameters

#### Input

*ld* Specifies the LDAP handle returned by a previous call to **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.

*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.

#### Output

*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 error message from the **LDAPMessage** message. The error message string should be freed by calling **ldap_memfree**. | NULL can be specified for *errmsgp* if the error message is to be ignored.

*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.

*resultoidp*
Specifies a pointer to an allocated, dotted-OID text string returned from the server. Free this string using the **ldap_memfree** API. If no OID is returned, **resultoidp* is set to NULL.

*resultdatap*
Specifies a pointer to a berval structure pointer that is set to an allocated copy of the data returned by the server. Free this struct berval using the code supplied in the **Notes** below. If no data is returned, **resultdatap* is set to NULL.

*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**.

*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.

The **ldap_parse_result**, **ldap_parse_extended_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 return **LDAP_SUCCESS** if the result was successfully located and parsed, and an LDAP error code if not successfully parsed.

The **ldap_err2string** API is used to convert the numeric LDAP error code, as returned by any of the LDAP APIs, into a NULL-terminated character string that describes the error. The character string is returned as static data and must not be freed by the application.

### Notes

These routines allocate storage. Use **ldap_memfree** to free the returned OID. Use the following code to free the returned struct berval:

```
if ( retdatap != NULL ) {
    if ( retdatap->bv_val != NULL ) {
        ldap_memfree( retdatap->bv_val );
    }
    ldap_memfree( (char *)retdatap );
}
```

## Error conditions

The parse APIs return an LDAP error code if they encounter an error parsing the result. See “ldap_error” on page 42 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_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 24 for more information about server controls.

*clientctrls*
Specifies a list of LDAP client controls. This parameter may be set to NULL. See "Supported client controls" on page 25 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 42 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 42 for possible values.

## 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_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 **LDAP_MSG_ONE**, search entries will be returned one at a time as they come in, through separate calls to **ldap_result**. If *all* is set to **LDAP_MSG_ALL**, the search response will only be returned in its entirety, that is, after all entries and the final search result have been received.

*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 87).

**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 50.

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

## Parameters

### Input

*ld* Specifies the LDAP handle returned by a previous call to **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> ::= '=' ; '~=' ; '<=' ; '>='
```

| **Note:** Use of the approximate filter (~=) is not supported on a z/OS LDAP server.

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 24 for more information about server controls.

*clientctrls*
Specifies a list of LDAP client controls. This parameter may be set to NULL. See “Supported client controls” on page 25 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`.

### Caching search results
Search results caching is supported. It can be enabled for specific LDAP connections using the **ldap_memcache** APIs, or globally for all connections by setting environment variables. See “ldap_memcache” on page 70 for details.

## 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 42 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 42 for possible values.

## Related topics

**ldap_result**
**ldap_error**
**ldap_memcache**

## ldap_server

**ldap_server_locate**
**ldap_server_conf_save**
**ldap_server_free_list**

### Purpose

Perform operations related to finding and saving published LDAP server information.

### Format

```
#include <ldap.h>
int ldap_server_locate (
       LDAPServerRequest *server_request,
       LDAPServerInfo **server_info_listpp);

int ldap_server_free_list(
        LDAPServerInfo *server_info_listp);

int ldap_server_conf_save(
       char *filename,
       unsigned long ttl,
       LDAPServerInfo *server_info_listp);

typedef struct LDAP_Server_Request {
       int search_source;           /* Source for server info     */
#define LDAP_LSI_CONF_DNS  0        /* Config first, then DNS (def)*/
#define LDAP_LSI_CONF_ONLY 1        /* Local Config file only     */
#define LDAP_LSI_DNS_ONLY  2        /* DNS only                   */
       char *conf_filename;         /* pathname of config file    */
       int reserved;                /* Reserved, set to zero      */
       char *service_key;           /* Service string             */
       char *enetwork_domain;       /* eNetwork domain (eDomain)  */
       char **name_servers;         /* Array of name server addrs */
       char **dns_domains;          /* Array of DNS domains       */
       int connection_type;         /* Connection type            */
#define LDAP_LSI_UDP_TCP 0          /* Use UDP, then TCP (default)*/
#define LDAP_LSI_UDP 1              /* Use UDP only               */
#define LDAP_LSI_TCP 2              /* Use TCP only               */
       int connection_timeout;      /* connect timeout (seconds)  */
       char *DN_filter;             /* DN naming context filter   */
       char *proto_key;             /* Symbolic protocol name     */
       unsigned char reserved2[60]; /* reserved fields, set to 0  */
} LDAPServerRequest;

typedef struct LDAP_Server_Info {
       char *lsi_host;           /* LDAP server's hostname */
       unsigned short lsi_port;  /* LDAP port              */
       char *lsi_suffix;         /* Server's LDAP naming context */
       char *lsi_query_key;      /* service_key[.edomain]  */
       char *lsi_dns_domain;     /* Publishing DNS domain  */
       int  lsi_replica_type;    /* master or replica      */
#define LDAP_LSI_MASTER  1       /* LDAP Master            */
#define LDAP_LSI_REPLICA 2       /* LDAP Replica           */
       int  lsi_sec_type;        /* SSL or non-SSL         */
#define LDAP_LSI_NOSSL   1       /* Non-SSL                */
#define LDAP_LSI_SSL     2       /* Secure Server          */
       unsigned short lsi_priority;   /* Server priority          */
       unsigned short lsi_weight;     /* load balancing weight    */
       char *lsi_vendor_info;         /* vendor information       */
       char *lsi_info;                /* LDAP Info string         */
       struct LDAP_Server_Info *prev; /* linked list previous ptr */
       struct LDAP_Server_Info *next; /* linked list next ptr     */
} LDAPServerInfo;
```

## Parameters

### Input

*server_request*
Specifies a pointer to an **LDAPServerRequest** structure, which should be initialized to zero prior to setting specific parameters. This will ensure that defaults are used when a parameter is not explicitly set. If the default behavior is desired for all possible input parameters, set *server_request* to NULL. This is equivalent to setting all elements of the **LDAPServerRequest** structure to zeroes. Otherwise, supply the address of the **LDAPServerRequest** structure, which contains the following fields:

*search_source*
Specifies where to find the server information. The options are:

- **LDAP_LSI_CONF_DNS**: First access the local LDAP DNS configuration file. If the file is not found, or the file does not contain information for a combination of the *service_key*, *enetwork_domain* and any of the DNS domains (as specified by the application), then access DNS.
- **LDAP_LSI_CONF_ONLY**: Search the local LDAP DNS configuration file only.
- **LDAP_LSI_DNS_ONLY**: Search DNS only.

*conf_filename*
Specifies an alternative configuration file name. Specify NULL to get the default file name and location (`/etc/ldap/ldap_server_info.conf`).

*service_key*
Specifies the search key (that is, the service name string) to be used when obtaining a list of SRV, “pseudo-SRV TXT” or CNAME alias records from DNS. If not specified, the default is **ldap**.

Note that standards are moving towards the use of **“_”** as a prefix for service name strings. Over time, it is expected that **“_ldap”** will be the preferred service name string for publishing LDAP services in DNS. If the application does not specify *service_key* and no entries are returned using the default **ldap** service name, the search will be automatically rerun using **“_ldap”** as the service name. As an alternative, the application can explicitly specify **“_ldap”** as the service name, and the search will be directed specifically at DNS SRV records that use **“_ldap”** as the service name.

Note that if *proto_key* is also unspecified, the default **“ldap”** *service_key* value is used in conjunction with the default **“tcp”** *proto_key* value. If a subsequent search is performed, the corresponding **“_ldap”** and **“_tcp”** default values are used.

*enetwork_domain*
Indicates that LDAP servers grouped within the specified eNetwork domain are to be located. An eNetwork domain is simply a naming construct, implemented by the LDAP administrator, to further subdivide a set of LDAP servers (as published in DNS) into logical groupings. By specifying an eNetwork domain, only the LDAP servers grouped within the specified eNetwork domain will be returned by the **ldap_server_locate** API. This can be very useful when an application, or group of applications, needs access to a particular set of LDAP servers. For example, the research division within a company might use a dedicated set of LDAP directories (masters and replicas). By publishing this set of LDAP servers in DNS with an eNetwork domain of “`research`”, applications that need to access to information published in research’s LDAP servers can selectively obtain the host names and ports of research’s LDAP servers. Other LDAP servers also published in DNS will not be returned.

The criteria for searching DNS to locate the appropriate LDAP server or servers is constructed by concatenating the following information:

- *service_key* (defaults to **ldap**)
- *enetwork_domain*
- *proto_key* (defaults to **tcp**)
- DNS domain

For example, if:

- The default **service_key** of **ldap** is used
- The eNetwork domain is `sales5`
- The client's default DNS domain is `midwest.acme.com`

then the DNS "value" used to search DNS for the set of LDAP servers belonging to the `sales5` eNetwork domain is `ldap.sales5.tcp.midwest.acme.com`.

If *enetwork_domain* is not specified, the following steps are taken to determine the *enetwork_domain*:

- The locally configured default, if set, is used (as set with the **ldap_enetwork_domain_set** API).
- If the locally configured default is not set, then the eNetwork domain component in the DNS name is omitted. In the above example, this would result in the following string being used: `ldap.tcp.midwest.acme.com`.

*name_servers*
Specifies a null-terminated array of DNS name server IP addresses (in dotted decimal format, for example, `122.122.33.49`). If not specified, the locally configured DNS name server or servers will be used.

*dns_domains*
Specifies a null-terminated array of one or more DNS domain names. If not specified, the local domain configuration is used.

Note that domain names supplied must be in standard DNS format. For example:

```
austin.ibm.com
```

**DNS domains and configuration file**

The local configuration file may contain server information for combinations of the following:
- Service key (typically set to **"ldap"** or **"_ldap"**)
- eNetwork domain
- DNS domains

When the application sets *search_source* to **LDAP_LSI_CONFIG_DNS** (the default), the **ldap_server_locate** API will attempt to find server information in the configuration file for the designated service key, eNetwork domain, and DNS domain or domains.

If the configuration file does not contain information that matches this criteria, the locator API will then search DNS, using the specified service key, eNetwork domain, and DNS domain or domains. For example:

- The application supplies the following three DNS domains:
  - `austin.ibm.com`
  - `raleigh.ibm.com`
  - `miami.ibm.com`

  Plus, the application uses the default service key (which is **ldap**) and specifies `sales` for the eNetwork domain).
- The configuration file contains server information for `austin.ibm.com` and `miami.ibm.com` (with the default service key and eNetwork domain of `sales`).
- Information is also published in DNS for `raleigh.ibm.com` (with the default service key and eNetwork domain of `sales`).
- The *search_source* parameter is set to **LDAP_LSI_CONFIG_DNS**, which indicates that both the configuration file and DNS are to be used if necessary.
- The locator API will build a single ordered list of server entries, with the following:
  - Server entries for the `austin.ibm.com` DNS domain, as extracted from the configuration file.

- Server entries for the `raleigh.ibm.com` DNS domain, as obtained from DNS over the network.
- Server entries for the `miami.ibm.com` DNS domain, as extracted from the configuration file.

In other words, the resulting list of servers will contain all the `austin.ibm.com` servers first, followed by the `raleigh.ibm.com` servers, followed by the `miami.ibm.com` servers. Within each grouping of servers (by DNS domain), the entries are sorted by priority and weight.

*connection_type*
Specifies the type of connection to use when communicating with the DNS name server. The following options are supported:
- **LDAP_LSI_UDP_TCP**: Use **UDP** first. If no response is received, or data truncation occurs, then use **TCP**.
- **LDAP_LSI_UDP**: Only use **UDP**.
- **LDAP_LSI_TCP**: Only use **TCP**.

UDP is the preferred connection type, and typically performs well. You might want to consider using TCP/IP if:
- The amount of data being returned will not fit in the 512-byte UDP packet.
- The transmission and receipt of UDP packets turns out to be unreliable. This may depend on network characteristics.

*connection_timeout*
Specifies a timeout value when querying DNS (for both **TCP** and **UDP**). If **LDAP_LSI_UDP_TCP** is specified for *connection_type* and a response is not received in the specified time period for **UDP**, **TCP** will be attempted. A value of zero results in an infinite timeout. When the **LDAPServerRequest** parameter is set to NULL, the default is ten seconds. When passing the **LDAPServerRequest** parameter, this parameter should be set to a non-zero value if an indefinite timeout is not desired.

*DN_filter*
Specifies a distinguished name (DN) to be used as a filter, for selecting candidate LDAP servers based on the server's naming context (or naming contexts). If the directory entry (as identified by the DN filter) has the potential to be contained within a directory hierarchy rooted by the queried naming context, an LDAPServerInfo structure is returned for the server/naming context combination. Best matching servers are returned first in the list.

*proto_key*
Specifies the protocol key (for example, **"tcp"** or **"_tcp"**) to be used when obtaining a list of SRV, "pseudo-SRV TXT" or CNAME alias records from DNS. If not specified, the default is **"tcp"**.

Note that standards are moving towards the use of "_" as a prefix for the protocol. Over time, it is expected that **"_tcp"** will be the preferred protocol string for publishing LDAP and other services in DNS. If the application does not specify *proto_key* and no entries are returned using the default **tcp** protocol key, the search will be automatically rerun using **"_tcp"** as the protocol. As an alternative, the application can explicitly specify **"_tcp"** as the protocol, and the search will be directly specifically at DNS SRV records that use **"_tcp"** as the protocol.

*reserved2*
Represents a reserved area for future function, which should be initialized to zero.

*server_info_listpp*
Specifies the address that will be set to point to a linked list of LDAPServerInfo structures. Each LDAPServerInfo structure defined in the list contains server information obtained from either:
- DNS
- Local configuration

**ldap_server**

*filename*
Specifies an alternative configuration file name. Specify NULL to get the default file name and location (`/etc/ldap/ldap_server_info.com`).

*ttl* Specifies the time-to-live (in minutes) for the server information saved in the configuration file. Set *ttl* to zero if it is intended to be a permanent repository of information.

When the **ldap_server_locate** API is used to access the configuration file with *search_source* set to **LDAP_LSI_CONF_ONLY**, and the configuration file has not been refreshed in *ttl* minutes, then **LDAP_TIMEOUT** error code is returned.

When the **ldap_server_locate** API is used to access the configuration file with *search_source* set to **LDAP_LSI_CONF_DNS**, and the configuration file has not been refreshed in *ttl* minutes, then network DNS is accessed to obtain server information.

*server_info_listp*
Specifies the address of a linked list of LDAPServerInfo structures. This linked list may have been returned from the **ldap_server_locate** API, or may be constructed by the application.

## Output

*server_info_listpp*
Upon successful return from **ldap_server_locate**, *server_info_listpp* points to a linked list of LDAPServerInfo structures. The LDAPServerInfo structure (as defined above), contains the following fields:

*lsi_host*
Fully-qualified hostname of the target server (NULL-terminated string).

*lsi_port*
Integer representation of the LDAP server's port.

*lsi_suffix*
String that specifies a supported naming context for the LDAP server (NULL-terminated string).

*lsi_query_key*
Specifies the eNetwork domain to which the LDAP server belongs, prefixed by the service key. For example, if *service_key* is **ldap** and *enetwork_domain* is `sales`, then *lsi_query_key* would be set to `ldap.sales`. If the server is not associated with an eNetwork domain (as published in DNS), then *lsi_query_key* consists solely of the service key value. For example, if the service key is **_ldap** and the eNetwork domain is not set, then *lsi_query_key* would be set to `_ldap.marketing`.

*lsi_dns_domain*
Specifies the DNS domain in which the LDAP server was published. For example, the DNS search may have been for `ldap.tcp.austin.ibm.com`, but the resulting server or servers has a fully-qualified DNS host name of `ldap2.raleigh.ibm.com`. In this example, *lsi_host* would be set to `ldap2.raleigh.ibm.com` while *lsi_dns_domain* would be set to `austin.ibm.com`. The actual domain in which the server was "published" may be of interest, particularly when multiple DNS domains are configured (or supplied as input).

*lsi_replica_type*
Specifies the type of server, **LDAP_LSI_MASTER** or **LDAP_LSI_REPLICA**. If set to zero, the type is unknown.

*lsi_sec_type*
Specifies the port's security type, **LDAP_LSI_NOSSL** or **LDAP_LSI_SSL**. This value is derived from the **ldap** or **ldaps** prefix on the LDAP URL returned. If the LDAP URL is not defined, the security type is unknown and *lsi_sec_type* is set to zero.

*lsi_priority*
Specifies the priority value obtained from the SRV RR (or the "pseudo-SRV" TXT RR). Set to zero if unknown or not available.

*lsi_weight*
Specifies the weight value obtained from the SRV RR (or the “pseudo-SRV” TXT RR). Set to zero if unknown or not available.

*lsi_vendor_info*
NULL-terminated string obtained from the ldapvendor TXT RR (if defined). May be used to identify the LDAP server vendor/version information.

*lsi_info*
NULL-terminated information string obtained from the ldapinfo TXT RR (if defined). If not defined, *lsi_info* is set to NULL. This information string can be used by the LDAP or network administrator to publish additional information about the target LDAP server.

*prev*
Points to the previous LDAP_Server_Info element in the linked list. This value is NULL if at the top of the list.

*next*
Points to the next LDAP_Server_Info element in the linked list. This value is NULL if at the end of the list.

## Usage

The **ldap_server_locate** API is used to locate one or more suitable LDAP servers. In general, an application will use the **ldap_server_locate** API, as follows:

- Prior to connecting to an LDAP server in the enterprise, use **ldap_server_locate** to obtain a list of one or more LDAP servers that have been published in DNS (or in the local configuration file). Typically an application can use the default request settings (by passing a NULL for the LDAPServerRequest parameter). By default, the API will look for server information in the local configuration file first, then move on to DNS if the local configuration file does not exist (or has expired).

  Note that if no server entries are found, and the application does not specify the service key (which defaults to **“ldap”**), then the **ldap_server_locate** function will re-run the complete search, using the alternative **“_ldap”** for the service key. The results of this second search, if any, will be returned to the application.
- Once the application has obtained the list of servers, it should walk the list, using the first server that meets its needs. This will maximize the advantage that can be derived from using the priority and weighting scheme implemented by the administrator. The application may not want to use the first server in the list for several reasons:
  - The client needs to specifically connect using SSL (or non-SSL). For each server in the list, the application can query the root DSE to determine if the server supports a secure SSL port (this is the preferred approach). For more information about accessing the root DSE, see “Searching a server’s root DSE” on page 140. Alternatively, the application can walk the list until it finds a server entry with the appropriate type of security. Note that an LDAP server may be listening on both an SSL and non-SSL port. In this case, the server will have two entries in the server list.
  - The client specifically needs to connect to a master (or replica).
  - The client needs to connect to a server that supports a particular naming context. Note that the list of servers returned in the list can be filtered by specifying *DN_filter*, which filters out servers that do not have a naming context under which the DN resides. To confirm that a server actually supports the naming context, it is recommended that the server’s root DSE be queried. See “Searching a server’s root DSE” on page 140 for more information.
  - There is some other characteristic associated with the desired server (possibly defined in the **ldapinfo** string).
- Once the client has selected a server, it then issues the **ldap_init** or **ldap_ssl_init** API. If the selected server is unavailable, the application is free to move down the list of servers until it either finds a suitable server it can connect to, or the list is exhausted.

### ldap_server

To free the list of servers (and associated LDAPServerInfo structures), the application should use the **ldap_server_free_list** API.

The **ldap_server_free_list** API is used to free the linked list of LDAPServerInfo structures (and all associated storage) as returned from the **ldap_server_locate** API.

The **ldap_server_conf_save** API is used to store server information into local configuration. The format for specifying the server information on the **ldap_server_conf_save** API is identical to the format returned from the **ldap_server_locate** API.

The application that writes information into the configuration file can specify an optional time-to-live for the information stored in the file. When an application uses the locator API to access DNS server information, the configuration file is considered to be stale if:

```
date/time_file_last_updated + ttl > current_date/time
```

If the application uses the default behavior for using the configuration file, it will bypass a stale configuration file and attempt to find all needed information from DNS. Otherwise, the *ttl* should be set to zero (indefinite *ttl*), in which case the information is considered to be good indefinitely.

Setting a non-zero *ttl* is most useful when an application (or other mechanism) exists for refreshing the local configuration file on a periodic basis.

Note that sub-second response time can be expected in many cases, when using UDP to query DNS. Since most applications will get the server information during initialization, repetitive invocation of the locator API is usually unnecessary.

By default, the configuration file is stored at the following location:

```
/etc/ldap/ldap_server_info.conf
```

## Format of local configuration file

Below is a sample definition for a local configuration file that is created with the **ldap_server_conf_save** API. It is recommended that the file be created with the **ldap_server_conf_save** API. However, with careful editing, it can also be created and maintained manually.

Some basic rules for managing this file manually:
- Comment fields are ignored, and must have “#” in the first character position
- All parameters are positional
- The first non-comment line must contain the time-to-live value for the file
- Contents of the file must be in IBM-1047 character set

```
######################################################################
# Local LDAP DNS configuration file.
#
# The first line holds the file's expiration time, which is
# a UNIX time_t value (time in seconds since January 1, 1970 UTC).
# A value of 0 indicates that the file will not expire.
#
# After the expiration time, Each of the following lines in
# this file represents a known LDAP server. The lines have
# the following format:
#
#  service domain host priority weight port replica sec "naming context" "vendor info" "general info"
#
# where:
#
#  service= service_key[.eNetwork_domain]
#
#  domain=   DNS domain
#
#  host=     fully qualified DNS name of the LDAP Server host
```

```
#
#  priority= target host with the lowest priority is tried first
#
#  weight=  load balancing method.  When multiple hosts have the
#           same priority, the host to be contacted first is determined
#           by the weight value.  Set to 0 if load balancing is not needed.
#
#  port=    The port to use to contact the LDAP Server.
#
#  replica= Use "1" to indicate Master.
#           "2" to indicate Replica.
#
#  sec=     Use "1" to indicate Non-SSL
#           "2" to indicate SSL.
#
#  naming context =  A naming context on the server.
#
#  vendor info= a string that identifies the LDAP server vendor
#
#  general info=    Any informational text you wish to include.
#
0
ldap austin.ibm.com ldapserver1.austin.ibm.com 1 1 389 1 1 "ou=users,o=ibm,c=us" "IBM SecureWay"
  "phoneinfo"
ldap austin.ibm.com ldapserver2.austin.ibm.com 1 1 389 2 1 "ou=users,o=ibm,c=us" "IBM SecureWay"
  "phoneinfo replica"
ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 636 1 2 "" ""
ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 636 1 2 "cn=GSO,o=IBM,c=US"
ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 636 1 2 "ou=Austin,o=IBM,c=US" "IBM" "GSO ePersonbase"
ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 389 1 1 "" ""
ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 389 1 1 "cn=GSO,o=IBM,c=US"
ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 389 1 1 "ou=Austin,o=IBM,c=US" "IBM" "GSO ePersonbase"
ldap.sales raleigh.ibm.com saleshost1.raleigh.ibm.com 1 1 389 1 1 "dc=raleigh,dc=ibm,dc=com"
  "IBM" "Sales Marketing"
ldap.sales raleigh.ibm.com saleshost2.raleigh.ibm.com 2 1 389 2 1 "dc=raleigh,dc=ibm,dc=com"
  "IBM" "Sales Marketing Replica"
######################################################################
```

The newer form of service keys can also be used in the configuration file. For example, the following is an excerpt that uses **"_ldap"** as the service key:

```
_ldap     austin.ibm.com ldapserver1.austin.ibm.com 1 1 389 1 1 "ou=users,o=ibm,c=us" "IBM SecureWay"
  "phoneinfo"
_ldap     austin.ibm.com ldapserver2.austin.ibm.com 1 1 389 2 1 "ou=users,o=ibm,c=us" "IBM SecureWay"
  "phoneinfo replica"
_ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 636 1 2 "" ""
_ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 636 1 2 "cn=GSO,o=IBM,c=US"
_ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 636 1 2 "ou=Austin,o=IBM,c=US" "IBM" "GSO ePersonbase"
_ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 389 1 1 "" ""
_ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 389 1 1 "cn=GSO,o=IBM,c=US"
_ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 389 1 1 "ou=Austin,o=IBM,c=US" "IBM" "GSO ePersonbase"
_ldap.sales raleigh.ibm.com saleshost1.raleigh.ibm.com 1 1 389 1 1 "dc=raleigh,dc=ibm,dc=com"
  "IBM" "Sales Marketing"
_ldap.sales raleigh.ibm.com saleshost2.raleigh.ibm.com 2 1 389 2 1 "dc=raleigh,dc=ibm,dc=com"
  "IBM" "Sales Marketing Replica"
```

## Publishing LDAP server information in DNS

If DNS is to be used to publish LDAP server information, the LDAP administrator must arrange to configure the relevant DNS name server or servers with the appropriate SRV and TXT records that reflect the LDAP servers available in the enterprise.

This includes:

- If SRV records are supported by the DNS servers in the enterprise (and their use is desired), SRV records must be created that identify the LDAP servers, along with appropriate weighting and priority settings. For more information on SRV records and how they are used, see IETF RFC 2052 *A DNS RR* *for specifying the location of services (DNS SRV)*, dated October 1996. A more recent draft describes a scheme where service keys and the protocol are prefixed with "_". See IETF RFC 2052 *A DNS RR for specifying the location of services (DNS SRV)*, dated January 1999, for more information on this proposed scheme.

- TXT records must be associated with the A record of each LDAP server published. The TXT records include the LDAP URL records (which specify host name, port, base DN and port type (**"ldap"** for non-SSL, and **"ldaps"** for SSL).
- If SRV records are not being used, the list of available servers must be specified with a set of TXT records (which emulate the SRV RR format).

The LDAP server locator API will:

- Provide access to a list of LDAP servers. By default, the locator API will query a local configuration file for the required information. If the file was updated with a nonzero time-to-live, and the file has become "stale", or the file does not contain the required information, the locator API will then access DNS. By default, the local configuration file has no time-to-live, and is considered to be good indefinitely. Note that the configuration file is designed to hold the same level of information per-server that can be obtained from DNS.
- Gather data relevant to each of the LDAP servers from DNS, using three algorithms:
  - SRV records
  - "pseudo-SRV" records (using TXT records)
  - A CNAME alias referencing a single host's A record

  The algorithms will be attempted in sequence until results are returned for one of the algorithms. For example, if no SRV records are found, but pseudo-SRV records are found, the list of servers is built from the pseudo-SRV records.
- Build a list of LDAP servers, with the first server in the list classified as the "preferred" or default server. Depending on how DNS is used to publish LDAP servers, the preferred LDAP server may actually be a reflection of how the administrator has organized the LDAP information in DNS. The application will have access to the additional data that was retrieved from DNS. The additional information for each LDAP server information structure can consist of the following:
  - Host name and port
  - eNetwork domain to which the server belongs
  - Fully-qualified DNS domain in which the host name is published
  - Naming context
  - Replication type (master or replica)
  - Security type (SSL or non-SSL)
  - Vendor ID
  - Administrator defined data

The application can use **ldap_server_locate** to obtain a list of one or more LDAP servers that exist in the enterprise, and have been published in DNS (or the local configuration file). The additional data may be used by the application to select the appropriate server. For example, the application may need a server that supports a specific naming context, or may need to specifically access the master for update operations.

As input to the API, the application can supply:

- A list of one or more DNS name server IP addresses. The default is to use the locally configured list of name server addresses. Once an active name server is located, it is used for all subsequent processing.
- The service key. The default is **"ldap"**. The service key is used to query DNS for information specific to the LDAP protocol. For example, when searching for SRV records in the `austin.ibm.com` DNS domain, the search would be for "`ldap.tcp.austin.ibm.com`" with type=SRV. This example assumes the search does not include an eNetwork domain component (see next item).

The application can also specify **"_ldap"** as the service key and **"_tcp"** for the protocol, in which case the search would be for "`_ldap._tcp.austin.ibm.com`" with type=SRV.

- The name of the eNetwork domain. The eNetwork domain is typically the name used to identify the LDAP user's authentication domain, and to further qualify the search for relevant LDAP servers, as published in the user's DNS domain. For example (to extend the previous example), when searching for SRV records in the `austin.ibm.com` DNS domain, with an eNetwork domain of marketing the search would be for "`ldap.marketing.tcp.austin.ibm.com`" with type=SRV.
- A list of one or more fully-qualified DNS domain names. The default is to use the locally configured domain or domains.

  If multiple domains are supplied (either in the default configuration, or explicitly supplied by the application), information is gathered from each DNS domain. The server information returned from the locator API is grouped by DNS domain. For example, if two domains are supplied (for example, `austin.ibm.com` and `raleigh.ibm.com`), the entries for LDAP servers published in the `austin.ibm.com` domain appear first in the list (with the `austin.ibm.com` servers sorted by priority and weight). Entries for LDAP servers published in the `raleigh.ibm.com` domain follow the entire set of `austin.ibm.com` servers (with the `raleigh.ibm.com` servers sorted by priority and weight). Note that all entries returned by the locator API are associated with a single *service_key.eNetwork_domain* combination.

  DNS domain names supplied here must be in the standard DNS format (for example, `austin.ibm.com`).
- The connection type (UDP or TCP).
- A DN for comparison against the naming context defined for each LDAP server entry. This string, if supplied, is used as a filter. Only server entries that define a naming context which compares with the DN are returned by the locator API. For example, a DN of "`cn=fred, ou=accounting, o=ibm, c=us`" matches the first of the following, but not the second:
  - `o=ibm, c=us`
  - `o=tivoli, c=us`

  The ability to filter based upon each LDAP server's naming context is supplied as a convenience, so the application does not need to step through the list of servers, comparing a DN with each entry's naming context.
- The application can specify how information residing in the local configuration file is used. The default is to look in the local configuration file for the desired information. If not found, then DNS servers on the network are accessed. The application can specify the following behaviors:
  - Look in configuration file first, then access network (default)
  - Look only in the configuration file
  - Access DNS only

  When using the default configuration file, the application does not need to specify the location. Alternatively, the application can provide a path name to a configuration file.

  Note that information stored in the configuration file takes the same form as information obtained from DNS. The difference is that it is saved in the file by an application. The file can also be constructed and distributed to end-users by the administrator.

Maximum benefit is obtained when applications can use the defaults for all the parameters (thus minimizing application knowledge of the specifics related to locating LDAP servers).

## Using SRV and TXT records

The DNS-lookup routine will look for SRV records first. If one or more servers are found, then the server information is returned and the second algorithm, based on TXT records that emulate SRV records, is not invoked.

The use of SRV records for finding the address of servers, for a specific protocol and domain, is described in IETF RFC 2052 *A DNS RR for Specifying the Location of Services (DNS SRV)*. Proper use of the SRV RR permits the administrator to distribute a service across multiple hosts within a domain, to move the service from host to host without disruption, as well as to designate certain hosts as primary and others as alternates, or backups (by using a priority and weighting scheme).

TXT stands for “TeXT”. TXT records are simply strings shared in a DNS server and associated with a DNS name. BIND versions prior to 4.8.3 do not support TXT records. To fully implement the technique described in RFC 2052, the DNS name servers must use a version of BIND that supports SRV records as well as TXT records. A SRV resource record (RR) has the following components (per RFC 2052):

```
service.proto.name ttl class SRV priority weight port target
```

where:

*service*
: Symbolic name of the desired service. By default, the service name (or service key) is **ldap**. When used to publish servers that are associated with an eNetwork domain, the service value is derived by concatenating the service key (for example, **ldap**) with the eNetwork domain name (for example, `marketing`). The resulting service would then be `ldap.marketing`.

*proto*
: Protocol, typically **tcp** or **udp** (or **_tcp** or **_udp**).

*name*
: Domain name associated with the RR.

*ttl*
: Time-to-live, standard DNS meaning.

*class*
: Standard DNS meaning (for example, IN).

*priority*
: Target host with lowest number priority should be attempted first.

*weight*
: Load balancing mechanism. When multiple target hosts have the same priority, the chance of contacting one of the hosts first should be proportional to its weight. (That is, a higher number makes it more likely the server will be contacted.) Set to 0 if load balancing is not necessary.

*port*
: Port on the target host for the service.

*target*
: Target host name (must have one or more A records associated with it).

The approach is to use SRV records to define a list of candidate LDAP servers, and to then use TXT records associated with each host’s A record to get additional information about each LDAP server. Three forms of TXT records are understood by the LDAP client DNS lookup routines:

```
ldap                A       199.23.45.296
                    TXT     "service:ldap://ldap.ibm.com:389/o=foo,c=us"
                    TXT     "ldaptype: master"
                    TXT     "ldapvendor: IBMeNetwork"
                    TXT     "ldapinfo: ldapver=3, keyx=fastserver"
```

The service TXT record provides a standard LDAP URL (provides host, port and base DN).

The ldaptype TXT record identifies whether the LDAP server is a master or replica.

The ldapvendor TXT record identifies the vendor.

The ldapinfo free-form TXT record provides additional information, as defined by the LDAP or network administrator. As in the example above, the information could be keyword based (the `ldapinfo` record is available to the application.

Finally, in combination, the name server configuration file should contain something like the following, which effectively publishes the set of LDAP servers that reside in the marketing eNetwork domain:

```
ldap.marketing.tcp    SRV     0  0  0    ldapm
                      SRV     0  0  0    ldapmsec
                      SRV     0  0  0    ldapmsuffix
                      SRV     1  1  0    ldapr1
                      SRV     1  2  0    ldapr2
                      SRV     1  2  0    ldapr2sec
                      SRV     2  1  2222 ldapr3.raleigh.ibm.com.

ldapm                 A       199.23.45.296
                      TXT     "service:ldap://ldapm.austin.ibm.com:389/o=foo,c=us"
                      TXT     "ldaptype: master"

ldapmsec              A       199.23.45.296
                      TXT     "service:ldaps://ldapm.austin.ibm.com:686/o=foo,c=us"
                      TXT     "ldaptype: master"
ldapmsuffix           A       199.23.45.296
                      TXT     "service:ldaps://ldapm.austin.ibm.com:389/o=moo,c=us"
                      TXT     "ldaptype: master"
ldapr1                A       199.23.45.297
                      TXT     "service:ldap://ldapr1:389/o=foo,c=us"
                      TXT     "ldaptype: replica"

ldapr2                A       199.23.45.298
                      TXT     "service:ldap://ldapr2:389/o=foo,c=us"
                      TXT     "ldaptype: replica"
ldapr2sec             A       199.23.45.298
                      TXT     "service:ldaps://ldapr2/o=foo,c=us"
                      TXT     "ldaptype: replica"
                      TXT     "ldapinfo: ca=verisign, authtype=server"

ldapr3.raleigh.ibm.com.   A   199.23.45.299
```

In this example, a DNS search for "`ibmldap.marketing.tcp.austin.ibm.com`" with type=SRV would return seven SRV records, which represent entries for four hosts. Note that a SRV record is needed for each port/naming context combination supported by a server. For example, a server that supports an SSL and non-SSL port, would have at least two SRV records, and two corresponding A records (that point to the same IP address). In this example, the A RR combinations for `ldapm/ldapmsec/ldapmsuffix` and `ldapr2/ldapr2sec` map to the same host address. Note that `ldapmsuffix` provides an alternate naming context for the 199.23.45.296 host.

The port specified on the SRV record is ignored if the target host has a TXT record containing an LDAP URL. If the URL is specified without a port, the default port is used (389 for non-SSL, 636 for SSL).

Some rules related to constructing the strings associated with the TXT records as they appear above:

- If the string contains white space, the entire string following the "TXT" must be enclosed in double quotes.
- If the string contains characters not supported by DNS (for example, the naming context might contain a character or characters not supported by DNS), an escape is supported, based on the technique described in IETF RFC 1738 *Uniform Resource Locators (URL)*. For example:

  ```
  TXT     "service:ldaps://ldapr2/o=foo%f0,c=us"
  ```

  permits the `x'f0'` character to be included in the LDAP URL.

The algorithm for the use of LDAP servers is outlined below. The LDAP servers are ordered in the list based on this algorithm. The application has the freedom of using the first server in the list (based on priority and weight). It also has the freedom to select a different server, based upon its needs.

## Using ″Pseudo-SRV″ TXT records

If the SRV algorithm does not return any servers, the secondary algorithm is invoked. Instead of looking for SRV records, the lookup routine will perform a TXT query using the service name string supplied on **ldap_server_locate**, which defaults to **ldap.tcp**.

**Note:** BIND versions prior to 8.x do not support “Pseudo-SRV” TXT records.

The intent with “Pseudo-SRV” records is to emulate the scheme provided with SRV records, but using a search for TXT records instead. To duplicate the previous example using TXT records instead of SRV records, the following definition (excerpt from a DNS server configuration file) is used:

```
ldap.marketing.tcp    TXT     0  0  0    ldapm
                      TXT     0  0  0    ldapmsec
                      TXT     0  0  0    ldapmsuffix
                      TXT     1  1  0    ldapr1
                      TXT     1  2  0    ldapr2
                      TXT     1  2  0    ldapr2sec
                      TXT     2  1  2222 ldapr3.raleigh.ibm.com.

ldapm                 A       199.23.45.296
                      TXT     "service:ldap://ldapm.austin.ibm.com:389/o=foo,c=us"
                      TXT     "ldaptype: master"

ldapmsec              A       199.23.45.296
                      TXT     "service:ldaps://ldapm.austin.ibm.com:686/o=foo,c=us"
                      TXT     "ldaptype: master"

ldapmsuffix           A       199.23.45.296
                      TXT     "service:ldaps://ldapm.austin.ibm.com:389/o=moo,c=us"
                      TXT     "ldaptype: master"

ldapr1                A       199.23.45.297
                      TXT     "service:ldap://ldapr1:389/o=foo,c=us"
                      TXT     "ldaptype: replica"

ldapr2                A       199.23.45.298
                      TXT     "service:ldap://ldapr2:389/o=foo,c=us"
                      TXT     "ldaptype: replica"

ldapr2sec             A       199.23.45.298
                      TXT     "service:ldaps://ldapr2/o=foo,c=us"
                      TXT     "ldaptype: replica"
                      TXT     "ldapinfo: ca=verisign, authtype=server"

ldapr3.raleigh.ibm.com.   A   199.23.45.299
```

The LDAP resolver routine will assume that the default domain is in effect when the “SRV-type TXT” records do not contain fully-qualified domain names.

Note that the pseudo-SRV TXT records, in many cases, can exactly replicate the syntax of SRV records, with the exception that “SRV” is replaced by “TXT”. However, some versions of DNS require data associated with the TXT records to be enclosed in double quotes, as follows:

```
ldap.marketing.tcp    TXT     "0  0  0    ldapm"
                      TXT     "0  0  0    ldapmsec"
```

The **ldap_server_locate** API handles either format.

## Using a CNAME alias record

If the pseudo-SRV algorithm does not return any servers, the third algorithm is invoked. Instead of looking for TXT records, the lookup routine will perform a standard query using the service name string supplied on **ldap_server_locate**, which defaults to **ldap**.

```
ldap.marketing.tcp    CNAME   ldapm

ldapm                 A       199.23.45.296
                      TXT     "service:ldap://ldapm.austin.ibm.com:389/o=foo,c=us"
                      TXT     "ldaptype: master"
```

If TXT records are not associated with the A record, defaults will be assumed for *port* and *ldaptype*.

## Alternative scheme for publishing LDAP server information in DNS

A more recent IETF draft describes a scheme where service keys and the protocol are prefixed with "_". See the following Internet draft for more information on this new scheme, which may obsolete RFC 2052: IETF RFC 2052 *A DNS RR for Specifying the Location of Services (DNS SRV)*, dated January 1999.

When services are published in DNS using the approach proposed in this IETF draft, service names and protocol are prefixed with "_".

For instance, a previous example would be defined as follows:

```
_ldap.marketing._tcp    SRV     0  0  0    ldapm
                        SRV     0  0  0    ldapmsec
                        SRV     0  0  0    ldapmsuffix
                        SRV     1  1  0    ldapr1
                        SRV     1  2  0    ldapr2
                        SRV     1  2  0    ldapr2sec
                        SRV     2  1  2222 ldapr3.raleigh.ibm.com.
```

If all LDAP service information is published within your enterprise in this way, the application can choose to not specify service key or protocol, and the **ldap_server_locate** API will first perform its search using **"ldap"** and **"tcp"**. The search will not find any entries, and the API will automatically rerun the search using **"_ldap"** and **"_tcp"** for service key and protocol, which will return the information published with the alternative scheme.

If information is published with both schemes, the application should explicitly define the service key and protocol, to ensure that the desired information is returned.

## ldap_server_locate usage by ldap_init, and ldap_ssl_init

The **ldap_init**, **ldap_open**, and **ldap_ssl_init** APIs are used to establish connections to LDAP servers. These APIs all accept a URL to identify the host and port of an LDAP server to communicate with. The format of the LDAP URL is as follows:

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

where:

| | |
|---|---|
| **ldap://** | Use an unsecured connection. |
| **ldaps://** | Use a Secure Socket Layer (SSL) secure connection. |
| *host* | An optional DNS-style host name of the LDAP server. |
| *port* | An optional port number. |
| *dn* | A distinguished name (DN). |

The *attributes*, *scope*, and *filter* portions of the URL are ignored by this operation.

When the *host* portion of the URL is omitted, these APIs internally call **ldap_server_locate** to locate an LDAP server to communicate with.

An LDAPServerRequest structure is mandatory input for all **ldap_server_locate** function calls, including the internal call made by these APIs. For this internal call, the fields in the input LDAPServerRequest structure are set as follows:

*DN_filter*
Set to the *dn* value specified in the URL (or NULL if unspecified).

*search_source*
Set to **LDAP_LSI_CONF_DNS**.

Default values are used for the remaining fields.

The fully-qualified file name identifying the local LDAP DNS configuration file to search is determined by the **LDAP_SERVER_INFO_CONF** environment variable setting. If this environment variable is not set, the default file (**/etc/ldap/ldap_server_info.conf**) is searched. If no matching LDAP server information is found in this file (or if the file is not found), then DNS is accessed to locate the server information.

If appropriate LDAP server information is returned by the internal call to **ldap_server_locate**, a connection is established to an LDAP server.

**Example:** Assume a local configuration file containing LDAP server information does not exist. Also, assume there is LDAP server information published in DNS identifying an LDAP server which:
- Serves the naming context "`o=IBM,c=US`"
- Listens on an unsecure (non-SSL) port

The following code generates an LDAP handle for communications with the LDAP server:

```
LDAP * ld = ldap_init( "ldap:///cn=Scott,o=IBM,c=US", 0 );      /* port parameter ignored */
```

The LDAP server defined to DNS which best matches the DN "`cn=Scott,o=IBM,c=US`" will be contacted by the LDAP client.

## Error conditions

The **ldap_server_locate**, **ldap_server_conf_save**, and **ldap_server_free_list** APIs return **LDAP_SUCCESS** if successful, otherwise an error code is returned. See "ldap_error" on page 42 for possible values.

## Related topics

**ldap_error**
**ldap_init**
**ldap_ssl**

## ldap_ssl

**ldap_ssl_client_init**
**ldap_ssl_init**
**ldap_ssl_start** (deprecated)

### Purpose

| Initialize the System Secure Socket Layer (SSL) and Transport Layer Security (TLS) functions for an LDAP
| application and create a secure (SSL/TLS) 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 *label)


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

### Parameters

**Input**

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

*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.

The *host* parameter can also be specified as a single LDAP URL. The format of the LDAP URL is:

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

where:

| | |
|---|---|
| *host* | Is an optional DNS-style host name. |
| *port* | Is an optional port number. |
| *dn* | Is the distinguished name. |

The *attributes*, *scope*, and *filter* portions of the URL are ignored by this operation.

The port number specified in the URL overrides the **ldap_ssl_init** parameter. If a port number is not specified in the URL, the default port 636 is used regardless of the URL format specified (**ldap** or **ldaps**).

If the URL host name is omitted, **ldap_ssl_init** attempts to locate an LDAP server to communicate with through an internal call to the **ldap_server_locate** function. In this case, the *dn* field (if specified) is used as input to **ldap_server_locate** to narrow the scope of eligible LDAP servers. See "ldap_server_locate usage by ldap_init, and ldap_ssl_init" on page 105 for details.

**Note:** A successful **ldap_ssl_init** operation will always initialize a secure SSL connection regardless of the URL format specified (**ldap** or **ldaps**).

*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 the System SSL key database file or RACF key ring. System SSL assumes that
| the name specifies a key database file. If the name is not a fully-qualified file name, then the current
| directory is assumed to contain the file. The key database file must be a file and cannot be an MVS
| dataset. If a corresponding file is not found then the name is assumed to specify a RACF key ring.

| See "Use of key databases and RACF key rings" on page 110 for more information on System SSL
| key databases and RACF key rings.

| **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).

| *keyring_pw*
| Specifies either the key database file password or the file specification for a System SSL password
| stash file. When the password stash file is used, it must be in the form **file://** followed immediately (no
| blanks) by the file specification (for example, `file:///etc/ldap/sslstashfile`). The stash file must be
| a file and cannot be an MVS dataset.

| *label*
Specifies the 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 *label* can be set to NULL). If the LDAP server is configured to perform client and server
| authentication, a client certificate is required. The *label* can be set to NULL if a certificate/private key
| pair has been designated as the default (using **gskkyman**). Similarly, *label* 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 154 for reason codes that can be returned.

## Usage

| The **ldap_ssl_client_init** API is used to initialize the SSL/TLS 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/TLS equivalent of **ldap_init**. It is used to initialize a secure 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 57 for more information about **ldap_get_option** and **ldap_set_option**.

| TCP/IP can cause a SIGPIPE signal to be generated when a peer closes their connection unexpectedly. In order for the TCP/IP function calls to be notified the SIGPIPE signal should be ignored. This causes an error return and EPIPE errno to be returned to the TCP/IP functions instead of creating the SIGPIPE signal. The application should code the signal ignore prior to invoking the **ldap_ssl_init** API. An example of the signal ignore call looks like:

```
sigignore(SIGPIPE);
```

Note that when connecting to an LDAP Version 2 server, the **ldap_simple_bind** call 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/TLS) to an LDAP server. The **ldap_ssl_start** API accepts the *ld* from an **ldap_open** and performs an SSL/TLS 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/TLS 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_init**, **ldap_ssl_start**, followed by **ldap_simple_bind**.

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

| | |
|---|---|
| LDAP_SSL_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_EX ″09″
| LDAP_SSL_3DES_SHA_US ″0A″
| LDAP_SSL_RSA_AES_128_SHA ″2F″
| LDAP_SSL_RSA_AES_256_SHA ″35″

## Use of key databases and RACF key rings

The **ldap_ssl** APIs can use either a System SSL key database or a RACF key ring. By obtaining certificates from trusted CAs, storing them in the key database file or RACF key ring, 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 or RACF key ring. 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/TLS 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 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 or key ring and mark it as trusted.

The contents of a client's System SSL key database file is managed with the **gskkyman** utility. See *z/OS: System Secure Sockets Layer Programming* for information about the **gskkyman** utility. The **gskkyman** utility is used to define the set of trusted CAs that are to be trusted by the client.

System SSL encrypts the key database file. Either the password must be specified or the file specification of a stash file that was created using the **gskkyman** utility must be specified in the form **file://** followed immediately (no blanks in between) by the file specification of the stash file.

RACF key rings are the recommended repository for certificates/keys. See the certificate/key management section in *z/OS: System Secure Sockets Layer Programming* for instructions on how to migrate a key database to RACF and how to use the RACDCERT command to protect the certificate and key ring.

The user ID under which the LDAP client runs must be authorized by RACF to use RACF key rings. To authorize the LDAP client, you can use the RACF commands in the following example (where *userid* is the user ID running the LDAP client utility):

```
RDEFINE  FACILITY  IRR.DIGTCERT.LIST  UACC(NONE)
RDEFINE  FACILITY  IRR.DIGTCERT.LISTRING  UACC(NONE)
PERMIT  IRR.DIGTCERT.LISTRING  CLASS(FACILITY)  ID(userid)  ACCESS(CONTROL)
PERMIT  IRR.DIGTCERT.LIST  CLASS(FACILITY)  ID(userid)  ACCESS(CONTROL)
```

Remember to refresh RACF after doing the authorizations.

```
SETROPTS  RACLIST(FACILITY)  REFRESH
```

See "ldap_init" on page 57 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/TLS session-level timer, use:

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

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

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**:

| LDAP_SSL_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_EX ″09″
| LDAP_SSL_3DES_SHA_US ″0A″
| LDAP_SSL_RSA_AES_128_SHA ″2F″
| LDAP_SSL_RSA_AES_256_SHA ″35″

| For more information on **ldap_set_option**, see "ldap_init" on page 57.

### 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/TLS versions of these utilities include RSA software.

## Related topics

**ldap_init**
**ldap_server**

## 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 */
        unsigned long lud_options; /* LDAP_URL_OPT_SECURE for "ldaps" format */
        } 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_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[s]://[host][:port][/dn][?attributes[?scope[?filter]]]
```

where:

- *host* 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>
```

### ldap_url

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:**

1. 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*.
2. If the LDAP URL does not contain a search filter, the filter defaults to `objectClass=*`.
3. Regarding the **ldaps://** form of the LDAP URL:
   - When input to APIs that establish connections to LDAP servers (**ldap_init**, and so on), this URL form indicates to use an SSL connection.
   - When input to **ldap_url_parse**, the *lud_options* field of the **LDAPURLDesc** structure returned is set to **LDAP_URL_OPT_SECURE**.

## 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 42 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 42 for possible values.

## Related topics

**ldap_search**

# Chapter 3. LDAP operation utilities

**Note:** This chapter does not contain programming interface information.

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 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 116 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: 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

| If you are using the utilities in interactive mode (for example, reading DNs, changetypes, and so on, from
| standard input), it is possible to break out of interactive mode by pressing <PA1>. This will return the TSO
| session to the READY prompt. This is similar to using <Control-C> in USS.

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 |

| z/OS shell name | TSO name |
|---|---|
| ldapsearch | ldapsrch |
| ldapdelete | ldapdlet |

## Using the command line utilities

The **ldapdelete**, **ldapmodify**, **ldapadd**, **ldapmodrdn**, and **ldapsearch** utilities all use the **ldap_bind_s** API. When bind is invoked, several results can be returned. Following are bind results using various combinations of user IDs and passwords.

1. If a null or 0 length DN is specified, the user receives unauthenticated access.
2. If a non-null, non-0 length DN is specified, a password must also be specified.
   - If the DN exists in the database, the entry must have a userpassword value and the password must match the specified value. The user is then bound with that identity.
   - If the DN does not exist in the database, the DN must be the administrator DN specified in the **adminDN** value in the configuration file and the password must match the **adminPW** value in the configuration file. The administrator DN must not be in the namespace of the LDAP server. The user is then bound as the administrator.

An error is returned when binding with any other combination of user ID and password.

**Note:** If you are using an LDAP server other than the z/OS LDAP server, the bind results may be different.

## SSL/TLS 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/TLS 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 the LDAP operation utilities without the **-Z** parameter and calling the secure port on an LDAP server (in other words, a non-secure call to a secure port) is not supported. Also, a secure call to a non-secure port is not supported.

| SSL/TLS encrypts the key ring file. Either the password must be specified as part of the **-P** parameter or file specification of a stash file that was created using the **gskkyman** utility must be specified in the form **file://** followed immediately (no blanks in between) by the file specification of the stash file.

## Using RACF key rings

Alternately, LDAP supports the use of a RACF key ring. See the certificate/key management section in *z/OS: System Secure Sockets Layer Programming* for instructions on how to migrate a key database to RACF and how to use the **RACDCERT** command to protect the certificate and key ring.

The user ID under which the LDAP client runs must be authorized by RACF to use RACF key rings. To authorize the LDAP client, you can use the RACF commands in the following example (where *userid* is the user ID running the LDAP client utility):

```
RDEFINE FACILITY IRR.DIGTCERT.LIST UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.LISTRING UACC(NONE)
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(userid) ACCESS(CONTROL)
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(userid) ACCESS(CONTROL)
```

Remember to refresh RACF after doing the authorizations.

```
SETROPTS RACLIST(FACILITY) REFRESH
```

Once the RACF key ring is set up and authorized, specify the RACF key ring name for the **-K** *keyfile* option and do not specify the **-P** *keyfilepw* option.

## CRAM-MD5 authentication to an IBM Directory Server

CRAM-MD5 authentication is supported on the IBM Directory Server and client utilities. However, the way that it has been implemented on the IBM Directory Server is different than the z/OS LDAP server. Thus, this has resulted in differences between the IBM Directory Server and the z/OS LDAP server client utilities. In order to perform a CRAM-MD5 authentication bind with the z/OS operation utilities to an IBM Directory Server, you must specify the bind DN (authorization DN) with the **-D** option. The **-U** *username* option on the z/OS operation utilities should not be used when attempting to do a CRAM-MD5 authentication bind to an IBM Directory Server because it 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 5. 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 **3**. |
| **-S** *method* or **-m** *method* | Specify the bind method to use. You can use either **-m** or **-S** to indicate the bind method.<br><br>The default is **SIMPLE**. You can also specify **GSSAPI** to indicate a Kerberos Version 5 is requested, **EXTERNAL** to indicate that a certificate (SASL external) bind is requested, **CRAM-MD5** to indicate that a SASL Challenge Response Authentication Mechanism bind is requested, or **DIGEST-MD5** to indicate a SASL digest hash bind is requested.<br><br>The **GSSAPI** method requires a protocol level of 3 and the user must have a valid Kerberos Ticket Granting Ticket in their credentials cache by using the Kerberos **kinit** command line utility.<br><br>The **EXTERNAL** method requires a protocol level of 3. 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.<br><br>The **CRAM-MD5** method requires a protocol level of 3. The **-D** or **-U** option must be specified.<br><br>The **DIGEST-MD5** method requires a protocol level of 3. The **-U** option must be specified. The **-D** option can optionally be used to specify the authorization DN. |
| **-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. |
| **-d** *debuglevel* | Specify the level of debug messages to be created. The debug level is specified in the same fashion as the debug level for the LDAP server. See Table 2 on page 16 for the possible values for *debuglevel*. The default is no debug messages. |

*Table 5. ldapdelete options (continued)*

| Option | Description |
|---|---|
| **-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.<br><br>If the **-S** or **-m** option is equal to **DIGEST-MD5** or **CRAM-MD5**, this option is the authorization DN which will be used for making access checks. This directive is optional when used in this manner. |
| **-w** *passwd* | Use *passwd* as the password for simple, **CRAM-MD5**, and **DIGEST-MD5** 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.<br><br>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: 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 non-secure port is 389 and the default LDAP secure port is 636. |
| **-Z** | Use a secure connection to communicate with the LDAP server. Secure connections expect the communication to begin with the SSL/TLS handshake.<br><br>The **-K** *keyfile* option or equivalent environment variable is required when the **-Z** option is specified. The **-P** *keyfilepw* option is required when the **-Z** option is specified and the key file specifies an HFS key database file. The **-N** *keyfilelabel* option must be specified if you wish to use a certificate that is different than the default specified in the key database. |
| **-K** *keyfile* | Specify the name of the System SSL key database file or RACF key ring. If this option is not specified, this utility looks for the presence of the **SSL_KEYRING** environment variable with an associated name.<br><br>System SSL assumes that the name specifies a key database file. If the name is not a fully-qualified file name, then the current directory is assumed to contain the file. The key database file must be a file and cannot be an MVS data set. If a corresponding file is not found then the name is assumed to specify a RACF key ring.<br><br>See “SSL/TLS information for LDAP utilities” on page 116 for information on System SSL key databases and RACF key rings.<br><br>This parameter is ignored if **-Z** is not specified. |
| **-P** *keyfilepw* | Specify either the key database file password or the file specification for a System SSL password stash file. When the stash file is used, it must be in the form **file://** followed immediately (no blanks) by the HFS file specification (for example, `file:///etc/ldap/sslstashfile`). The stash file must be a file and cannot be an MVS data set.<br><br>This parameter is ignored if **-Z** is not specified. |
| **-N** *keyfilelabel* | Specify the label associated with the key in the System SSL key database or RACF key ring.<br><br>This parameter is ignored if **-Z** is not specified |
| **-U** *userName* | Specify the user name for **CRAM-MD5** or **DIGEST-MD5** binds. The *userName* is a short name (for example, the **uid** attribute value) that will be used to perform bind authentication.<br><br>This option is required if the **-S** or **-m** option is set to **DIGEST-MD5**. |

Table 5. ldapdelete options (continued)

| Option | Description |
|---|---|
| **-g** *realmName* | Specify the realm name to use when doing a **DIGEST-MD5** bind. This option is required when multiple realms are passed from an LDAP server to a client as part of a **DIGEST-MD5** challenge; otherwise, it is optional. |

**-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

Following are some **ldapdelete** 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).
- For z/OS LDAP support for RACF access, the following command:

  ```
  ldapdelete -D racfid=admin1,profiletype=user,sysplex=sysplexa -w passwd
    "racfid=u1,profiletype=user,sysplex=sysplexa"
  ```

  attempts to delete the RACF user `u1` and remove all the connections of `u1` to RACF groups. It is assumed that the z/OS LDAP support for RACF access suffix is `sysplex=sysplexa` and that `admin1` has the RACF authority to make this update to RACF.

## 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>.

The **LDAP_DEBUG** environment variable may be used to set the debug level. For more information on specifying the debug level using keywords, decimal, hexadecimal, and plus and minus syntax, see “Tracing” on page 15.

If you are attempting a CRAM-MD5 authentication bind to an IBM Directory Server, see “CRAM-MD5 authentication to an IBM Directory Server” on page 117 for more information.

You can specify an LDAP URL for *ldaphost* on the **-h** parameter. See 58 for more information.

## SSL/TLS note

See “SSL/TLS information for LDAP utilities” on page 116.

## 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 6. 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 **3**. |
| **-S** *method* or **-m** *method* | Specify the bind method to use. You can use either **-m** or **-S** to indicate the bind method.<br><br>The default is **SIMPLE**. You can also specify **GSSAPI** to indicate a Kerberos Version 5 is requested, **EXTERNAL** to indicate that a certificate (SASL external) bind is requested, **CRAM-MD5** to indicate that a SASL Challenge Response Authentication Mechanism bind is requested, or **DIGEST-MD5** to indicate a SASL digest hash bind is requested.<br><br>The **GSSAPI** method requires a protocol level of 3 and the user must have a valid Kerberos Ticket Granting Ticket in their credentials cache by using the Kerberos **kinit** command line utility.<br><br>The **EXTERNAL** method requires a protocol level of 3. 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.<br><br>The **CRAM-MD5** method requires a protocol level of 3. The **-D** or **-U** option must be specified.<br><br>The **DIGEST-MD5** method requires a protocol level of 3. The **-U** option must be specified. The **-D** option can optionally be used to specify the authorization DN. |
| **-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. |
| **-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. |

Table 6. ldapmodify and ldapadd options (continued)

| Option | Description |
|---|---|
| **-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* | Specify the level of debug messages to be created. The debug level is specified in the same fashion as the debug level for the LDAP server. See Table 2 on page 16 for the possible decimal values for *debuglevel*. The default is no debug messages. |
| **-f** *file* | Read the entry modification information from *file* instead of from standard input.<br><br>If you specify an LDIF file as input for this option, it must be an HFS file (and not a dataset). In addition, the separator line between LDIF clauses must be blank (that is, it must not contain any characters or white space). |
| **-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.<br><br>If the **-S** or **-m** option is equal to **DIGEST-MD5** or **CRAM-MD5**, this option is the authorization DN which will be used for making access checks. This directive is optional when used in this manner. |
| **-w** *passwd* | Use *passwd* as the password for simple, **CRAM-MD5**, and **DIGEST-MD5** 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.<br><br>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: 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 non-secure port is 389 and the default LDAP secure port is 636. |
| **-Z** | Use a secure connection to communicate with the LDAP server. Secure connections expect the communication to begin with the SSL/TLS handshake.<br><br>The **-K** *keyfile* option or equivalent environment variable is required when the **-Z** option is specified. The **-P** *keyfilepw* option is required when the **-Z** option is specified and the key file specifies an HFS key database file. The **-N** *keyfilelabel* option must be specified if you wish to use a certificate that is different than the default specified in the key database. |
| **-K** *keyfile* | Specify the name of the System SSL key database file or RACF key ring. If this option is not specified, this utility looks for the presence of the **SSL_KEYRING** environment variable with an associated name.<br><br>System SSL assumes that the name specifies a key database file. If the name is not a fully-qualified file name, then the current directory is assumed to contain the file. The key database file must be a file and cannot be an MVS data set. If a corresponding file is not found then the name is assumed to specify a RACF key ring.<br><br>See "SSL/TLS information for LDAP utilities" on page 116 for information on System SSL key databases and RACF key rings.<br><br>This parameter is ignored if **-Z** is not specified.<br><br>Once the RACF key ring is set up and authorized, specify the RACF key ring name for the **-K** *keyfile* option and do not specify the **-P** *keyfilepw* option. |

*Table 6. ldapmodify and ldapadd options (continued)*

| Option | Description |
|---|---|
| **-P** *keyfilepw* | Specify either the key database file password or the file specification for a System SSL password stash file. When the stash file is used, it must be in the form **file://** followed immediately (no blanks) by the HFS file specification (for example, `file:///etc/ldap/sslstashfile`). The stash file must be a file and cannot be an MVS data set.<br><br>This parameter is ignored if **-Z** is not specified. |
| **-N** *keyfilelabel* | Specify the label associated with the key in the System SSL key database or RACF key ring.<br><br>This parameter is ignored if **-Z** is not specified |
| **-U** *userName* | Specify the user name for **CRAM-MD5** or **DIGEST-MD5** binds. The *userName* is a short name (for example, the **uid** attribute value) that will be used to perform bind authentication.<br><br>This option is required if the **-S** or **-m** option is set to **DIGEST-MD5.** |
| **-g** *realmName* | Specify the realm name to use when doing a **DIGEST-MD5** bind. This option is required when multiple realms are passed from an LDAP server to a client as part of a **DIGEST-MD5** challenge; otherwise, it is optional. |

## 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:

## ldapmodify and ldapadd

```
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}
[{attrtype}{sep}{value}]
.
.
.
-
```

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. Specifying **delete** without any attribute-value pair records removes all the values for the specified attribute. Specifying **delete** followed by one or more attribute-value pair records removes only those values specified in the attribute-value pair records.

If an **add:x**, **replace:x**, or **delete:***attrtype* 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.

**LDIF mode examples*:*** Here are some examples of valid input for the **ldapmodify** command using LDIF mode.

*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
```

### ldapmodify and ldapadd

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.
| The new RDN, `cn=Tim Tom Doe`, is added automatically by the LDAP server to the values of the **cn**
| attribute in the entry.

*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: tim@yourcompany.com
registeredaddress: timtd@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: telephonenumber
-
delete: registeredaddress
registeredaddress: tim@yourcompany.com
-
```

This example deletes the **telephonenumber** attribute, deletes a single **registeredaddress** attribute value, and adds a **description** attribute.

*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`.

***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**. For a delete modification the equal sign (=) and *value* should be omitted to remove an entire attribute. 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.

**Modify mode examples*:*** Here are some examples of valid input for the **ldapmodify** command using modify mode.

*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: tim@yourcompany.com
registeredaddress: timtd@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.

### ldapmodify and ldapadd

*Deleting an attribute type:*

```
cn=Tim Doe, ou=Your Department, o=Your Company, c=US
-registeredaddress=tim@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.

## Examples

| Following are some **ldapmodify** and **ldapadd** examples. In these examples, replace the bind DN (*binddn*)
| and bind password (*bindpw*) with an identity with appropriate authority for your installation.

- 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`, adds 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`.

- Assuming that the file **/tmp/schemamods** exists and has the contents:

  ```
  dn: cn=schema, o=My Company, c=US
  -attributetypes=( 1.2.1 NAME 'attr1' DESC 'attribute type' \
    EQUALITY caseIgnoreMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
  +attributetypes=( 1.2.1 NAME 'attr1' DESC 'attribute type - obsoleted' OBSOLETE \
    EQUALITY caseIgnoreMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
  +attributetypes=( 1.2.2 NAME 'attr2' DESC 'new attribute type' \
    EQUALITY caseIgnoreMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
  +ibmattributetypes=( 1.2.2 ACCESS-CLASS normal )
  -objectclasses=( 4.5.6 NAME 'oc1' DESC 'sample object class' STRUCTURAL MUST ( cn ) )
  +objectclasses=( 4.5.6 NAME 'oc1' DESC 'sample object class' STRUCTURAL MUST ( cn ) MAY ( attr2 ) )
  ```

  the command:

**ldapmodify and ldapadd**

```
ldapmodify -f /tmp/schemamods -h hostA -D binddn -w bindpw
```

obsoletes the `attr1` attribute type definition by specifying the **OBSOLETE** keyword in the definition, adds the `attr2` attribute type definition and the associated IBM attribute type information, and modifies the `oc1` object class definition by adding the `attr2` attribute type as a **MAY** attribute.

- Assuming that the file **/tmp/newentry** exists and has the contents:

```
dn: racfid=u1,profiletype=user,sysplex=sysplexa
objectclass: racfuser
objectclass: racfbasecommon
racfid: u1
racfdefaultgroup: racfid=g1,profiletype=group,sysplex=sysplexa
racfconnectgroupUACC: read
racfconnectgroupauthority: join
```

the command:

```
ldapadd -D racfid=admin1,profiletype=user,sysplex=sysplexa -w bindpw -f /tmp/newentry
```

creates a RACF user named `u1`, with join authority and update UACC in the group `g1`. It is assumed that the z/OS LDAP support for RACF access suffix is `sysplex=sysplexa` and that `admin1` has the RACF authority to make this update to RACF.

In the following LDIF, the x on the replace: x line is a placeholder for the attribute name and allows multiple attribute names and values to be replaced in a single operation. If the file **/tmp/modentry** contains:

```
dn: racfid=u1,profiletype=user,sysplex=sysplexa
changetype: modify
replace: x
racfattributes: OPERATIONS
racfconnectgroupUACC: update
```

the command:

```
ldapmodify -D racfid=admin1,profiletype=user,sysplex=sysplexa -w bindpw -f /tmp/modentry
```

adds the **racfattributes** attribute to `OPERATIONS` and changes the **racfconnectgroupUACC** to `update`.

## Notes

The **LDAP_DEBUG** environment variable may be used to set the debug level. For more information on specifying the debug level using keywords, decimal, hexadecimal, and plus and minus syntax, see "Tracing" on page 15.

If you are attempting a CRAM-MD5 authentication bind to an IBM Directory Server, see "CRAM-MD5 authentication to an IBM Directory Server" on page 117 for more information.

You can specify an LDAP URL for *ldaphost* on the **-h** parameter. See 58 for more information.

## SSL/TLS note

See "SSL/TLS information for LDAP utilities" on page 116.

## 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_rename** 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 being renamed may be either leaf entries or
| non-leaf entries, and entire subtrees may be relocated in the directory with the command-line **-s** option.

The **ldapmodrdn** utility is not supported by z/OS LDAP support for RACF access.

### Format

```
ldapmodrdn [options] {-f file |  < entryfile | dn newrdn }
```

### Parameters

*options*
The following table shows the *options* you can use for the **ldapmodrdn** utility:

*Table 7. 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 **3**. |
| **-S** *method* or **-m** *method* | Specify the bind method to use. You can use either **-m** or **-S** to indicate the bind method.<br><br>The default is **SIMPLE**. You can also specify **GSSAPI** to indicate a Kerberos Version 5 is requested, **EXTERNAL** to indicate that a certificate (SASL external) bind is requested, **CRAM-MD5** to indicate that a SASL Challenge Response Authentication Mechanism bind is requested, or **DIGEST-MD5** to indicate a SASL digest hash bind is requested.<br><br>The **GSSAPI** method requires a protocol level of 3 and the user must have a valid Kerberos Ticket Granting Ticket in their credentials cache by using the Kerberos **kinit** command line utility.<br><br>The **EXTERNAL** method requires a protocol level of 3. 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.<br><br>The **CRAM-MD5** method requires a protocol level of 3. The **-D** or **-U** option must be specified.<br><br>The **DIGEST-MD5** method requires a protocol level of 3. The **-U** option must be specified. The **-D** option can optionally be used to specify the authorization DN. |
| **-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. |

Table 7. ldapmodrdn options (continued)

| Option | Description |
|---|---|
| **-d** *debuglevel* | Specify the level of debug messages to be created. The debug level is specified in the same fashion as the debug level for the LDAP server. See Table 2 on page 16 for the possible values for *debuglevel*. The default is no debug messages. |
| **-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.<br><br>If the **-S** or **-m** option is equal to **DIGEST-MD5** or **CRAM-MD5**, this option is the authorization DN which will be used for making access checks. This directive is optional when used in this manner. |
| **-w** *passwd* | Use *passwd* as the password for simple, **CRAM-MD5**, and **DIGEST-MD5** 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.<br><br>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: 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 non-secure port is 389 and the default LDAP secure port is 636. |
| **-Z** | Use a secure connection to communicate with the LDAP server. Secure connections expect the communication to begin with the SSL/TLS handshake.<br><br>The **-K** *keyfile* option or equivalent environment variable is required when the **-Z** option is specified. The **-P** *keyfilepw* option is required when the **-Z** option is specified and the key file specifies an HFS key database file. The **-N** *keyfilelabel* option must be specified if you wish to use a certificate that is different than the default specified in the key database. |
| **-K** *keyfile* | Specify the name of the System SSL key database file or RACF key ring. If this option is not specified, this utility looks for the presence of the **SSL_KEYRING** environment variable with an associated name.<br><br>System SSL assumes that the name specifies a key database file. If the name is not a fully-qualified file name, then the current directory is assumed to contain the file. The key database file must be a file and cannot be an MVS data set. If a corresponding file is not found then the name is assumed to specify a RACF key ring.<br><br>See “SSL/TLS information for LDAP utilities” on page 116 for information on System SSL key databases and RACF key rings.<br><br>This parameter is ignored if **-Z** is not specified. |
| **-P** *keyfilepw* | Specify either the key database file password or the file specification for a System SSL password stash file. When the stash file is used, it must be in the form **file://** followed immediately (no blanks) by the HFS file specification (for example, `file:///etc/ldap/sslstashfile`). The stash file must be a file and cannot be an MVS data set.<br><br>This parameter is ignored if **-Z** is not specified. |
| **-N** *keyfilelabel* | Specify the label associated with the key in the System SSL key database or RACF key ring. |
| **-U** *userName* | Specify the user name for **CRAM-MD5** or **DIGEST-MD5** binds. The *userName* is a short name (for example, the **uid** attribute value) that will be used to perform bind authentication.<br><br>This option is required if the **-S** or **-m** option is set to **DIGEST-MD5**. |

*Table 7. ldapmodrdn options (continued)*

| Option | Description |
|---|---|
| **-g** *realmName* | Specify the realm name to use when doing a **DIGEST-MD5** bind. This option is required when multiple realms are passed from an LDAP server to a client as part of a **DIGEST-MD5** challenge; otherwise, it is optional. |
| **-s** *newSuperior* | Specify the DN of the new superior entry under which the renamed entry will be relocated. The *newSuperior* argument may be the zero-length string (-s “”), if the destination server accepts zero-length string *newSuperior* arguments on an LDAP Modify DN operations. |
| **-l** *seconds* | Sends an **IBMModifyDNTimelimitControl** control with the operation request, substituting *seconds* as the control value. The control criticality is set to **TRUE**. Refer to *z/OS: Security Server LDAP Server Administration and Use* for a description of this control. |
| **-a** | Sends an **IBMModifyDNRealignDNAttributesControl** control with the operation request. The control criticality is set to **TRUE**. There is no control value. Refer to *z/OS: Security Server LDAP Server Administration and Use* for a description of this control. |

**-f** *file*
: Read the entry rename information from *file* instead of from standard input or the command line (by specifying *dn* and *newrdn*). Only the first pair of *dn* and *newrdn* values will be read from the file. All others will be ignored. The *newSuperior* option may not be included in *file*; this option is only accepted as a command-line option.

*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 a single pair of lines. The first line indicates the DN and the second line indicates the RDN.

## Examples

Following is an **ldapmodrdn** example:

- 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
  ```

  changes the RDN from `cn=Modify Me, o=My Company, c=US` to `cn=The New Me` and removes the old RDN `cn=Modify Me, o=My Company, c=US`.

- The command:

  ```
  ldapmodrdn -r -l 30 "cn=Modify Me, o=My Company, c=US" "cn=The New Me"
  ```

  changes the RDN from `cn=Modify Me, o=My Company, c=US` to `cn=The New Me` and removes the old RDN `cn=Modify Me, o=My Company, c=US`. An **IBMModifyDNTimelimitControl** control will accompany the operation request, specifying a time limit of 30 seconds.

- The command:

  ```
  ldapmodrdn -l 30 -a -s "o=Some Other Company, c=US" "cn=Modify Me, o=My Company, c=US' "cn=The New Me"
  ```

changes the RDN from `cn=Modify Me, o=My Company, c=US` to `cn=The New Me` and removes the old RDN `cn=Modify Me, o=My Company, c=US`.

The renamed entry will be relocated beneath the new superior entry `o=Some Other Company, c=US`. If the renamed entry is a non-leaf node, its subordinate entries will also be moved and renamed to reflect their new locations in the directory hierarchy. An **IBMModifyDNTimelimitControl** control will accompany the operation request, specifying a time limit of 30 seconds, and an **IBMModifyDNRealignDNAttributesControl** control will accompany the operation request.

## Notes

The **LDAP_DEBUG** environment variable may be used to set the debug level. For more information on specifying the debug level using keywords, decimal, hexadecimal, and plus and minus syntax, see “Tracing” on page 15.

You can specify an LDAP URL for *ldaphost* on the **-h** parameter. See 58 for more information.

If you are attempting a CRAM-MD5 authentication bind to an IBM Directory Server, see “CRAM-MD5 authentication to an IBM Directory Server” on page 117 for more information.

For clients using authenticated binds, the DNs in their identity mappings may change as a result of a Modify DN operation which is performed concurrently with their session to the server, and this may affect ACL processing which results in permission to access, or denial of access to, directory entries for which they previously were permitted or denied access. The resolution for this situation is to unbind and rebind so that identity processing uses the latest DNs.

## SSL/TLS note

See “SSL/TLS information for LDAP utilities” on page 116.

## 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 LDAP Server.

### Format

```
ldapsearch [options] filter [attributes...]
```

### Parameters

*options*

The following table shows the *options* you can use for the **ldapsearch** utility:

*Table 8. 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 **3**. |
| **-S** *method* or **-m** *method* | Specify the bind method to use. You can use either **-m** or **-S** to indicate the bind method.<br><br>The default is **SIMPLE**. You can also specify **GSSAPI** to indicate a Kerberos Version 5 is requested, **EXTERNAL** to indicate that a certificate (SASL external) bind is requested, **CRAM-MD5** to indicate that a SASL Challenge Response Authentication Mechanism bind is requested, or **DIGEST-MD5** to indicate a SASL digest hash bind is requested.<br><br>The **GSSAPI** method requires a protocol level of 3 and the user must have a valid Kerberos Ticket Granting Ticket in their credentials cache by using the Kerberos **kinit** command line utility.<br><br>The **EXTERNAL** method requires a protocol level of 3. 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.<br><br>The **CRAM-MD5** method requires a protocol level of 3. The **-D** or **-U** option must be specified.<br><br>The **DIGEST-MD5** method requires a protocol level of 3. The **-U** option must be specified. The **-D** option can optionally be used to specify the authorization DN. |
| **-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. |

Table 8. ldapsearch options (continued)

| Option | Description |
|---|---|
| **-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. The **LANG** environment variable must be set appropriately in the shell so that the desired characters print. Note that the default **LANG** value of **C** causes the desired characters not to print. |
| **-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. |
| **-d** *debuglevel* | Specify the level of debug messages to be created. The debug level is specified in the same fashion as the debug level for the LDAP server. See Table 2 on page 16 for the possible decimal values for *debuglevel*. The default is no debug messages. |
| **-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. |
| **-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.<br><br>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* for more information.<br><br>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:<br>• 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.<br>• If the client has not passed a limit, and has bound as the **adminDN**, then the limit will be considered unlimited.<br>• 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. |

*Table 8. ldapsearch options (continued)*

| Option | Description |
|---|---|
| **-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:<br>• 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.<br>• If the client has not passed a limit, and has bound as the **adminDN**, then the limit will be considered unlimited.<br>• 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.<br>• When accessing the z/OS LDAP support for RACF, the number of entries returned is limited by the z/OS LDAP server. See the information about accessing RACF information in *z/OS: Security Server LDAP Server Administration and Use*. |
| **-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.<br><br>If the **-S** or **-m** option is equal to **DIGEST-MD5** or **CRAM-MD5**, this option is the authorization DN which will be used for making access checks. This directive is optional when used in this manner. |
| **-w** *passwd* | Use *bindpasswd* as the password for simple, CRAM-MD5, and DIGEST-MD5 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.<br><br>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: 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 non-secure port is 389 and the default LDAP secure port is 636. |
| **-Z** | Use a secure connection to communicate with the LDAP server. Secure connections expect the communication to begin with the SSL/TLS handshake.<br><br>The **-K** *keyfile* option or equivalent environment variable is required when the **-Z** option is specified. The **-P** *keyfilepw* option is required when the **-Z** option is specified and the key file specifies an HFS key database file. The **-N** *keyfilelabel* option must be specified if you wish to use a certificate that is different than the default specified in the key database. |
| **-K** *keyfile* | Specify the name of the System SSL key database file or RACF key ring. If this option is not specified, this utility looks for the presence of the **SSL_KEYRING** environment variable with an associated name.<br><br>System SSL assumes that the name specifies a key database file. If the name is not a fully-qualified file name, then the current directory is assumed to contain the file. The key database file must be a file and cannot be an MVS data set. If a corresponding file is not found then the name is assumed to specify a RACF key ring.<br><br>See "SSL/TLS information for LDAP utilities" on page 116 for information on System SSL key databases and RACF key rings.<br><br>This parameter is ignored if **-Z** is not specified. |

*Table 8. ldapsearch options (continued)*

| Option | Description |
|---|---|
| **-P** *keyfilepw* | Specify either the key database file password or the file specification for a System SSL password stash file. When the stash file is used, it must be in the form **file://** followed immediately (no blanks) by the HFS file specification (for example, `file:///etc/ldap/sslstashfile`). The stash file must be a file and cannot be an MVS data set.<br><br>This parameter is ignored if **-Z** is not specified. |
| **-N** *keyfiledn* | Specify the label associated with the key in the System SSL key database or RACF key ring. |
| **-U** *userName* | Specify the user name for CRAM-MD5 or DIGEST-MD5 binds. The *userName* is a short name (for example, the **uid** attribute value) that will be used to perform bind authentication.<br><br>This option is required if the **-S** or **-m** option is set to **DIGEST-MD5**. |
| **-g** *realmName* | Specify the realm name to use when doing a **DIGEST-MD5** bind. This option is required when multiple realms are passed from an LDAP server to a client as part of a **DIGEST-MD5** challenge; otherwise, it is optional. |

*filter*
: Specify an IETF RFC 1558 compliant LDAP search filter. (See “ldap_search” on page 87 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. Each example makes the assumption that the LDAP server is running on the local host and listening on the default LDAP port (389).

- The command:

  ```
  ldapsearch -b "o=IBM University,c=US" "cn=karen smith" cn telephoneNumber
  ```

  performs a subtree search using the search base `"o=IBM University,c=US"` 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, 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, o=IBM University, c=US
  ```

```
cn=Karen Smith
cn=Karen Diane Smith
cn=Karen D Smith
telephoneNumber=+1 313 555-2277
```

- The command:

  ```
  ldapsearch -b "o=IBM University,c=US" -t "uid=kds" jpegPhoto audio
  ```

  performs a subtree search using the search base "`o=IBM University,c=US`" 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, 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 -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 -D racfid=admin1,profiletype=user,sysplex=sysplexa -w passwd
    -b "profiletype=user,sysplex=sysplexa" "racfid=G*"
  ```

**ldapsearch**

performs a search in the `user` subtree of the z/OS LDAP support for RACF access for the RACF users whose names begin with `G`. Only the DN of each matching entry is displayed. The z/OS LDAP support for RACF access suffix is assumed to be `sysplex=sysplexa`. The output might look like:

```
racfid=G\#126,profiletype=USER,sysplex=sysplexa
racfid=GDCEBLD,profiletype=USER,sysplex=sysplexa
racfid=GKUPERM,profiletype=USER,sysplex=sysplexa
racfid=GLDSRV,profiletype=USER,sysplex=sysplexa
...
```

To then retrieve the entire entry for one of the matching users, use the command:

```
ldapsearch -D racfid=admin1,profiletype=user,sysplex=sysplexa -w passwd
  -b "racfid=gkuperm,profiletype=user,sysplex=sysplexa" "objectclass=*"
```

The results might look like:

```
racfid=GKUPERM,profiletype=USER,sysplex=sysplexa
objectclass=racfUser
objectclass=racfBaseCommon
racfid=GKUPERM
racfprogrammername=UNKNOWN
racfowner=racfid=SUSET1,profiletype=USER,sysplex=sysplexa
racfauthorizationdate=01.017
racfdefaultgroup=racfid=SYS1,profiletype=GROUP,sysplex=sysplexa
racfpasswordchangedate=00.000
racfpasswordinterval=186
racfattributes=NONE
racfrevokedate=NONE
racfresumedate=NONE
...
```

## Searching a server's root DSE

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: Security Server LDAP Server Administration and Use* for more information about root DSE and what the z/OS LDAP server returns.

## Notes

The **LDAP_DEBUG** environment variable may be used to set the debug level. For more information on specifying the debug level using keywords, decimal, hexadecimal, and plus and minus syntax, see "Tracing" on page 15.

| If you are attempting a CRAM-MD5 authentication bind to an IBM Directory Server see page 117 for more
| information.

You can specify an LDAP URL for *ldaphost* on the **-h** parameter. See page 58 for more information.

## SSL/TLS note

See "SSL/TLS information for LDAP utilities" on page 116.

## 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.

# 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
 * 5694-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 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
 * 5694-A01
 * (C) Copyright IBM Corp. 1997, 2002
 *
 */

/*
 * 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

#define LDAP_URL_PREFIX         "ldap://"
#define LDAPS_URL_PREFIX        "ldaps://"

/* 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)

/* valid inputs for all flag of ldap_result */
#define LDAP_MSG_ONE            0x00    /* return one result */
#define LDAP_MSG_ALL            0x01    /* return all results */

/* authentication methods available */
#define LDAP_AUTH_SIMPLE        0x80L   /* context specific+primitive */
#define LDAP_AUTH_SASL_30       0xa3L   /* context specific+constructed */
#define LDAP_AUTH_SASL          0xa3L   /* context specific+constructed */


/* search scopes */
#define LDAP_SCOPE_BASE     0x00
#define LDAP_SCOPE_ONELEVEL 0x01
#define LDAP_SCOPE_SUBTREE  0x02

/* bind constants */
#define LDAP_MECHANISM_EXTERNAL "EXTERNAL"
#define LDAP_MECHANISM_EXTERNAL_UTF8 "\x45\x58\x54\x45\x52\x4E\x41\x4C"
/* Kerberos V5 Mechanism */
#define LDAP_MECHANISM_GSSAPI   "GSSAPI"
#define LDAP_MECHANISM_GSSAPI_UTF8 "\x47\x53\x53\x41\x50\x49"
/* CRAM-MD5 Mechanism */
#define LDAP_MECHANISM_CRAM   "CRAM-MD5"
#define LDAP_MECHANISM_CRAM_UTF8 "\x43\x52\x41\x4D\x2D\x4D\x44\x35"
/* DIGEST-MD5 Mechanism */
#define LDAP_MECHANISM_DIGEST   "DIGEST-MD5"
#define LDAP_MECHANISM_DIGEST_UTF8 "\x44\x49\x47\x45\x53\x54\x2D\x4D\x44\x35"

#define LDAP_SASL_SIMPLE        ""

/* 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;
```

```
/* Server Request Search Source Types */
#define LDAP_LSI_CONF_DNS   0   /* Config first, then DNS (def) */
#define LDAP_LSI_CONF_ONLY  1   /* Ccnfiguration file only      */
#define LDAP_LSI_DNS_ONLY   2   /* DNS only                     */

/* Server Request Connection Types */
#define LDAP_LSI_UDP_TCP    0   /* UDP first, then TCP */
#define LDAP_LSI_UDP        1   /* Use UDP only */
#define LDAP_LSI_TCP        2   /* Use TCP only */

/* LDAP Server Types */
#define LDAP_LSI_MASTER     1   /* LDAP Master */
#define LDAP_LSI_REPLICA    2   /* LDAP Replica */

/* LDAP Server Security Types */
#define LDAP_LSI_NOSSL      1   /* Non-SSL */
#define LDAP_LSI_SSL        2   /* Secure Server */

/*
 * options that can be set/gotten
 */
#define LDAP_OPT_SIZELIMIT          0x00
#define LDAP_OPT_TIMELIMIT          0x01
#define LDAP_OPT_REFERRALS          0x02
#define LDAP_OPT_DEREF              0x03
#define LDAP_OPT_RESTART            0x04
#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
#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_DELEGATION         0xE3
/*
 * option value to indicate that one-time (per process)
 * SSL initialization should be avoided since the initialization
 * was already done elsewhere in the application. (deprecated)
 */
#define LDAP_OPT_SSL_AVOIDSTATICINIT 0xE4 /* deprecated */
#define LDAP_OPT_DEBUG_STRING       0xE5


#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
```

```
#define LDAP_OPT_V2_WIRE_FORMAT_ISO8859_1   0x00
#define LDAP_OPT_V2_WIRE_FORMAT_UTF8        0x01

/* 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_BE_CAPABILITIES  0x00400000
#define LDAP_DEBUG_CACHE        0x00800000
#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"        /* deprecated */
#define LDAP_SSL_DES_SHA_EX     "09"
#define LDAP_SSL_3DES_SHA_US    "0A"
#define LDAP_SSL_RSA_AES_128_SHA  "2F"
#define LDAP_SSL_RSA_AES_256_SHA  "35"
#define LDAP_SSL_CIPHERLIST     "03060504090A352F"

/*
 * possible error codes we can return.      hex value          decimal value
 */

#define LDAP_SUCCESS                        0x00              /*    0 */
#define LDAP_OPERATIONS_ERROR               0x01              /*    1 */
#define LDAP_PROTOCOL_ERROR                 0x02              /*    2 */
#define LDAP_TIMELIMIT_EXCEEDED             0x03              /*    3 */
#define LDAP_SIZELIMIT_EXCEEDED             0x04              /*    4 */
#define LDAP_COMPARE_FALSE                  0x05              /*    5 */
#define LDAP_COMPARE_TRUE                   0x06              /*    6 */
#define LDAP_STRONG_AUTH_NOT_SUPPORTED      0x07              /*    7 */
#define LDAP_STRONG_AUTH_REQUIRED           0x08              /*    8 */
#define LDAP_PARTIAL_RESULTS                0x09              /*    9 */
```

```
#define LDAP_REFERRAL                     0X0a            /*  10 */
#define LDAP_ADMIN_LIMIT_EXCEEDED         0X0b            /*  11 */
#define LDAP_UNAVAILABLE_CRITICAL_EXTENSION 0X0c          /*  12 */
#define LDAP_CONFIDENTIALITY_REQUIRED     0x0d            /*  13 */
#define LDAP_SASLBIND_IN_PROGRESS         0x0e            /*  14 */

#define LDAP_NO_SUCH_ATTRIBUTE            0x10            /*  16 */
#define LDAP_UNDEFINED_TYPE               0x11            /*  17 */
#define LDAP_INAPPROPRIATE_MATCHING       0x12            /*  18 */
#define LDAP_CONSTRAINT_VIOLATION         0x13            /*  19 */
#define LDAP_TYPE_OR_VALUE_EXISTS         0x14            /*  20 */
#define LDAP_INVALID_SYNTAX               0x15            /*  21 */

#define LDAP_NO_SUCH_OBJECT               0x20            /*  32 */
#define LDAP_ALIAS_PROBLEM                0x21            /*  33 */
#define LDAP_INVALID_DN_SYNTAX            0x22            /*  34 */
#define LDAP_IS_LEAF                      0x23            /*  35 */
#define LDAP_ALIAS_DEREF_PROBLEM          0x24            /*  36 */

#define LDAP_INAPPROPRIATE_AUTH           0x30            /*  48 */
#define LDAP_INVALID_CREDENTIALS          0x31            /*  49 */
#define LDAP_INSUFFICIENT_ACCESS          0x32            /*  50 */
#define LDAP_BUSY                         0x33            /*  51 */
#define LDAP_UNAVAILABLE                  0x34            /*  52 */
#define LDAP_UNWILLING_TO_PERFORM         0x35            /*  53 */
#define LDAP_LOOP_DETECT                  0x36            /*  54 */

#define LDAP_NAMING_VIOLATION             0x40            /*  64 */
#define LDAP_OBJECT_CLASS_VIOLATION       0x41            /*  65 */
#define LDAP_NOT_ALLOWED_ON_NONLEAF       0x42            /*  66 */
#define LDAP_NOT_ALLOWED_ON_RDN           0x43            /*  67 */
#define LDAP_ALREADY_EXISTS               0x44            /*  68 */
#define LDAP_NO_OBJECT_CLASS_MODS         0x45            /*  69 */
#define LDAP_RESULTS_TOO_LARGE            0x46            /*  70 */

#define LDAP_AFFECTS_MULTIPLE_DSAS        0X47            /*  71 */

#define LDAP_OTHER                        0x50            /*  80 */
#define LDAP_SERVER_DOWN                  0x51            /*  81 */
#define LDAP_LOCAL_ERROR                  0x52            /*  82 */
#define LDAP_ENCODING_ERROR               0x53            /*  83 */
#define LDAP_DECODING_ERROR               0x54            /*  84 */
#define LDAP_TIMEOUT                      0x55            /*  85 */
#define LDAP_AUTH_UNKNOWN                 0x56            /*  86 */
#define LDAP_FILTER_ERROR                 0x57            /*  87 */
#define LDAP_USER_CANCELLED               0x58            /*  88 */
#define LDAP_PARAM_ERROR                  0x59            /*  89 */
#define LDAP_NO_MEMORY                    0x5a            /*  90 */
#define LDAP_CONNECT_ERROR                0x5b            /*  91 */
#define LDAP_NOT_SUPPORTED                0x5c            /*  92 */
#define LDAP_CONTROL_NOT_FOUND            0x5d            /*  93 */
#define LDAP_NO_RESULTS_RETURNED          0x5e            /*  94 */
#define LDAP_MORE_RESULTS_TO_RETURN       0x5f            /*  95 */

#define LDAP_URL_ERR_NOTLDAP              0x60            /*  96 */
#define LDAP_URL_ERR_NODN                 0x61            /*  97 */
#define LDAP_URL_ERR_BADSCOPE             0x62            /*  98 */
#define LDAP_URL_ERR_MEM                  0x63            /*  99 */

#define LDAP_CLIENT_LOOP                  0x64            /* 100 */
#define LDAP_REFERRAL_LIMIT_EXCEEDED      0x65            /* 101 */

#define LDAP_SSL_ALREADY_INITIALIZED      0x70            /* 112 */
#define LDAP_SSL_INITIALIZE_FAILED        0x71            /* 113 */
#define LDAP_SSL_CLIENT_INIT_NOT_CALLED   0x72            /* 114 */
#define LDAP_SSL_PARAM_ERROR              0x73            /* 115 */
```

```
#define LDAP_SSL_HANDSHAKE_FAILED          0x74              /*  116 */
#define LDAP_SSL_GET_CIPHER_FAILED         0x75              /*  117 */
#define LDAP_SSL_NOT_AVAILABLE             0x76              /*  118 */

#define LDAP_NO_EXPLICIT_OWNER             0x80              /*  128 */
#define LDAP_NO_EXPLICIT_ACL               0x81              /*  129 */

#define LDAP_DNS_NO_SERVERS                0xC5              /*  197 */
#define LDAP_DNS_TRUNCATED                 0xC6              /*  198 */
#define LDAP_DNS_INVALID_DATA              0xC7              /*  199 */
#define LDAP_DNS_RESOLVE_ERROR             0xC8              /*  200 */
#define LDAP_DNS_CONF_FILE_ERROR           0xC9              /*  201 */


/*
 * 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 */
#define LDAP_URL_OPT_SECURE     2
        unsigned long lud_options;
    } 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;

/*
```

```
 * structure for LDAP Server Request
 */

    typedef struct LDAP_Server_Request {
        int search_source;              /* Search DNS, Cache or both */
        char    *conf_filename;         /* Configuratin file */
        int reserved;                   /* Reserved, set to 0 */
        char    *service_key;           /* Service identifier */
        char    *enetwork_domain;       /* eNetwork domain name */
        char    **name_servers;         /* Array of name servers */
        char    **dns_domains;          /* Array of DNS domains */
        int connection_type;            /* Use UDP, TCP or both */
        int connection_timeout;         /* Connection timeout in seconds */
        char *DN_filter;                /* DN suffix filter */
        char *proto_key;                /* Symbolic protocol name */
        unsigned char reserved2[60];    /* Reserved, set to 0 */
    } LDAPServerRequest;

/*
 * structure for LDAP Server Information
 */

    typedef struct LDAP_Server_Info {
        char    *lsi_host;              /* LDAP server's hostname */
        unsigned short lsi_port;        /* LDAP port  */
        char    *lsi_suffix;            /* server suffix */
        char    *lsi_query_key;         /* service_key[.enetwork_domain] */
        char    *lsi_dns_domain;        /* publishing DNS domain */
        int lsi_replica_type;           /* master or replica */
        int lsi_sec_type;               /* SSL or non-SSL */
        unsigned short lsi_priority;    /* server priority */
        unsigned short lsi_weight;      /* load balancing weight */
        char    *lsi_vendor_info;       /* vendor information    */
        char    *lsi_info;              /* LDAP info string */
        struct LDAP_Server_Info *prev;  /* linked list previous pointer */
        struct LDAP_Server_Info *next;  /* linked lsit next pointer */
    } LDAPServerInfo;

/* 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 );
    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);
    int ldap_extended_operation(LDAP *ld, const char* exoid,
                                const struct berval* exdata,
                                LDAPControl **serverctrls,
                                LDAPControl** clientctrls, int* msgidp);
    int ldap_extended_operation_s(LDAP *ld, const char* exoid,
                                  const struct berval* exdata,
                                  LDAPControl **serverctrls,
                                  LDAPControl** clientctrls, char **retoidp,
                                  struct berval **retdatap);


    int ldap_server_locate (LDAPServerRequest *server_request,
                            LDAPServerInfo **server_info_listpp);
    int ldap_server_free_list (LDAPServerInfo *server_info_listp);
    int ldap_server_conf_save (char *basename, unsigned long ttl,
                               LDAPServerInfo *server_info_listp);

    int ldap_enetwork_domain_set (char *name, char *filename);
    int ldap_enetwork_domain_get (char **name, char *filename);


/*
 * client-side search entry cache functions
 */
    typedef struct ClientCache LDAPMemCache;

/*
 * The following definition of ldap_thread_fns is provided for
 * compilation compatibility with the caching APIs defined by
 * other SDKs.  This structure (if specified) is NOT USED by the
 * z/OS implementation; all cache access serialization is performed
 * internally by the cache code itself.
 */

    typedef void *(LDAP_TF_MUTEX_ALLOC_CALLBACK)( void );
    typedef void (LDAP_TF_MUTEX_FREE_CALLBACK)( void * );
    typedef int (LDAP_TF_MUTEX_LOCK_CALLBACK)( void * );
    typedef int (LDAP_TF_MUTEX_UNLOCK_CALLBACK)( void * );
    typedef int (LDAP_TF_GET_ERRNO_CALLBACK)( void );
    typedef void (LDAP_TF_SET_ERRNO_CALLBACK)( int );
    typedef int (LDAP_TF_GET_LDERRNO_CALLBACK)( char **, char **, void * );
    typedef void (LDAP_TF_SET_LDERRNO_CALLBACK)( int, char *, char *,
                                                 void * );
    typedef struct _ldap_thread_fns {
        LDAP_TF_MUTEX_ALLOC_CALLBACK *ltf_mutex_alloc;
        LDAP_TF_MUTEX_FREE_CALLBACK *ltf_mutex_free;
        LDAP_TF_MUTEX_LOCK_CALLBACK *ltf_mutex_lock;
        LDAP_TF_MUTEX_UNLOCK_CALLBACK *ltf_mutex_unlock;
        LDAP_TF_GET_ERRNO_CALLBACK *ltf_get_errno;
        LDAP_TF_SET_ERRNO_CALLBACK *ltf_set_errno;
        LDAP_TF_GET_LDERRNO_CALLBACK *ltf_get_lderrno;
        LDAP_TF_SET_LDERRNO_CALLBACK *ltf_set_lderrno;
        void *ltf_lderrno_arg;
    } ldap_thread_fns ;

    void ldap_memcache_update( LDAPMemCache *cache );
    int ldap_memcache_set( LDAP *ld, LDAPMemCache *cache );
    int ldap_memcache_init( unsigned long ttl, unsigned long size,
                            char **baseDNs, ldap_thread_fns *reserved,
                            LDAPMemCache **cachep );
    int ldap_memcache_get( LDAP *ld, LDAPMemCache **cachep );
    void ldap_memcache_flush( LDAPMemCache *cache, char *dn, int scope );
    void ldap_memcache_destroy( LDAPMemCache *cache );


/*
 * Numeric OIDs for supported CLIENT controls
 */

/*
 * ibm-serverHandledSearchRequest client control numeric oid
 */
#define IBM_SERVER_HANDLED_SEARCH_REQUEST_OID      "1.3.18.0.2.10.7"
/*
 * ibm-serverHandledSearchRequest client control numeric oid represented in
 * UTF-8 (for use with the LDAP_OPT_UTF8_IO setting)
 */
#define IBM_SERVER_HANDLED_SEARCH_REQUEST_OID_UTF8                      \
        "\x31\x2E\x33\x2E\x31\x38\x2E\x30\x2E\x32\x2E\x31\x30\x2E\x37"


/*
 * ibm-saslBindDigestRealmName and ibm-saslBindCramRealmName
 * client control numeric oid
 */
#define IBM_CLIENT_MD5_REALM_NAME_OID       "1.3.18.0.2.10.12"

/*
 * ibm-saslBindDigestRealmName and ibm-saslBindCramRealmName
 * client control numeric oid represented in
 * UTF-8 (for use with the LDAP_OPT_UTF8_IO setting)
 */
#define IBM_CLIENT_MD5_REALM_NAME_OID_UTF8                               \
        "\x31\x2E\x33\x2E\x31\x38\x2E\x30\x2E\x32\x2E\x31\x30\x2E\x31\x32"


/*
 * ibm-saslBindDigestUserName and ibm-saslBindCramUserName
 * client control numeric oid
 */
#define IBM_CLIENT_MD5_USER_NAME_OID       "1.3.18.0.2.10.13"

/*
 * ibm-saslBindDigestUserName and ibm-saslBindCramUserName
 * client control numeric oid represented in
 * UTF-8 (for use with the LDAP_OPT_UTF8_IO setting)
 */
#define IBM_CLIENT_MD5_USER_NAME_OID_UTF8                               \
        "\x31\x2E\x33\x2E\x31\x38\x2E\x30\x2E\x32\x2E\x31\x30\x2E\x31\x33"
```

```
/* IBMModifyDNTimelimitControl OID  */
#define MODDN_TIMELIMIT_OID_STR       "1.3.18.0.2.10.10"

/*
 * IBMModifyDNTimelimitControl OID
 * client control numeric oid represented in
 * UTF-8 (for use with the LDAP_OPT_UTF8_IO setting)
 */
#define MODDN_TIMELIMIT_OID_STR_UTF8                              \
        "\x31\x2E\x33\x2E\x31\x38\x2E\x30\x2E\x32\x2E\x31\x30\x2E\x31\x30"

/* IBMModifyDNRealignDNAttributesControl OID  */
#define MODDN_REALIGN_DNATTR_OID_STR  "1.3.18.0.2.10.11"

/*
 * IBMModifyDNRealignDNAttributesControl OID
 * client control numeric oid represented in
 * UTF-8 (for use with the LDAP_OPT_UTF8_IO setting)
 */
#define MODDN_REALIGN_DNATTR_OID_STR_UTF8                              \
        "\x31\x2E\x33\x2E\x31\x38\x2E\x30\x2E\x32\x2E\x31\x30\x2E\x31\x31"

/*
 * BER encoded TRUE and FALSE BOOLEAN values.  These can be used
 * to specify the ldctl_value.bv_val for client and server controls
 * accepting a BER encoded BOOLEAN value (for example: the
 * ibm-serverHandledSearchRequest client control)
 */
#define BER_ENCODED_BOOLEAN_FALSE "\x30\x03\x01\x01\x00"
#define BER_ENCODED_BOOLEAN_TRUE  "\x30\x03\x01\x01\xFF"



#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
 * 5694-A01
 * (C) Copyright IBM Corp. 1997, 2002
 *
 */

#ifndef _LDAPSSL_H
#define _LDAPSSL_H

/*
    SSL Reason Codes.  The #defines in this header map the
    System SSL return codes to LDAP defined reason codes.

    The following table documents the mapping between the SSL return codes
    and the LDAP reason codes.  Comments on the #define lines provide
    brief descriptions of the error.  In addition, SSL manuals provide
    additional descriptions of the SSL return codes.

    Note: LDAP reason code -99 is used to map unrecognized SSL return codes.
          In this case, recreate the problem with LDAP Debug set to error.
          A Debug error statement will indicate the SSL return code.
    Note: In the table, "n/a" for the SSL Return Code Value indicates that
          LDAP Reason is not applicable to a specific SSL return code.  SSL
          did not return an error that caused the LDAP reason code.

     LDAP Reason   SSL Return
     Code Value    Code Value   LDAP Reason Code Name
     -----------   ----------   ---------------------------------------------
           -1         402       LDAP_SSL_ERROR_NO_CIPHERS
           -2         403       LDAP_SSL_ERROR_NO_CERTIFICATE
           -6         405       LDAP_SSL_ERROR_UNSUPPORTED_CERTIFICATE_TYPE
          -10         406       LDAP_SSL_ERROR_IO
          -11         410       LDAP_SSL_ERROR_BAD_MESSAGE
          -12         411       LDAP_SSL_ERROR_BAD_MAC
          -13         412       LDAP_SSL_ERROR_UNSUPPORTED
          -14         413       LDAP_SSL_ERROR_BAD_CERT_SIG
          -15         414       LDAP_SSL_ERROR_BAD_CERT
          -16         415       LDAP_SSL_ERROR_BAD_PEER
          -17         416       LDAP_SSL_ERROR_PERMISSION_DENIED
          -18         417       LDAP_SSL_ERROR_SELF_SIGNED
          -20           4       LDAP_SSL_ERROR_BAD_MALLOC
          -21           5       LDAP_SSL_ERROR_BAD_STATE
          -22         420       LDAP_SSL_ERROR_SOCKET_CLOSED
          -40         421       LDAP_SSL_SOC_BAD_V2_CIPHER
          -41         422       LDAP_SSL_SOC_BAD_V3_CIPHER
          -99      12 or any    LDAP_SSL_ERROR_UNKNOWN_ERROR
                 other unmapped
                   SSL rc
```

```
        -1000         n/a       LDAP_SSL_DLL_LOAD_FAILED
        -1001         n/a       LDAP_SSL_ROUTINE_MISSING
        -1002         n/a       LDAP_SSL_LDAP_LOCKINIT_FAILED
            1         102       LDAP_SSL_KEYFILE_IO_ERROR
            2         202       LDAP_SSL_KEYFILE_OPEN_FAILED
            4         408       LDAP_SSL_KEYFILE_BAD_PASSWORD
           12        6, 407     LDAP_SSL_KEYFILE_BAD_LABEL
          106         106       LDAP_SSL_BAD_FORMAT_OR_INVALID_PASSWORD
          109         109       LDAP_SSL_KEYFILE_NO_CA_CERTIFICATES
          201         201       LDAP_SSL_NO_KEYFILE_PASSWORD
          203         203       LDAP_SSL_RSA_TEMP_KEY_PAIR
          204         204       LDAP_SSL_KEYFILE_PASSWORD_EXPIRED
          301         301       LDAP_SSL_CLOSE_FAILED
          302         302       LDAP_SSL_CONNECTION_ACTIVE
          401         401       LDAP_SSL_ERR_BAD_DATE
          427         427       LDAP_SSL_ERR_LDAP_NOT_AVAILABLE
          428         428       LDAP_SSL_ERR_NO_PRIVATE_KEY
          429         429       LDAP_SSL_ERR_INVALID_V2_HEADER
          432         432       LDAP_SSL_ERR_NO_NEGOTIATION
          433         433       LDAP_SSL_ERR_EXPORT_RESTRICTION
          434         434       LDAP_SSL_ERR_INCOMPATIBLE_KEY
          435         435       LDAP_SSL_ERR_UNKNOWN_CA
          436         436       LDAP_SSL_ERR_BAD_CRL
          437         437       LDAP_SSL_ERR_CONNECTION_CLOSED
          438         438       LDAP_SSL_ERR_INTERNAL_ERROR_ALERT
          439         439       LDAP_SSL_ERR_UNKNOWN_ALERT
          501         501       LDAP_SSL_INVALID_BUFFER_SIZE
          502         502       LDAP_SSL_WOULD_BLOCK
          503         503       LDAP_SSL_WOULD_BLOCK_READ
          504         504       LDAP_SSL_WOULD_BLOCK_WRITE
          505         505       LDAP_SSL_ERR_RECORD_OVERFLOW
          601         601       LDAP_SSL_ERR_NOT_SSLV3
          602         602       LDAP_SSL_MISC_INVALID_ID
          701         701       LDAP_SSL_ATTRIBUTE_INVALID_ID
          702         702       LDAP_SSL_ATTRIBUTE_INVALID_LENGTH
          704         704       LDAP_SSL_ATTRIBUTE_INVALID_SID_CACHE
          703         703       LDAP_SSL_ATTRIBUTE_INVALID_ENUMERATION
          705         705       LDAP_SSL_ATTRIBUTE_INVALID_NUMERIC_VALUE
          706         706       LDAP_SSL_ATTRIBUTE_INVALID_PARAMETER
        10001           1       LDAP_SSL_INVALID_HANDLE
        10003           3       LDAP_SSL_INTERNAL_ERROR
        10007           7       LDAP_SSL_CERTIFICATE_NOT_AVAILABLE
        10008           8       LDAP_SSL_CERT_VALIDATION
        10009           9       LDAP_SSL_ERR_CRYPTO
        10010          10       LDAP_SSL_ERR_ASN
        10011          11       LDAP_SSL_ERR_LDAP
        10103         103       LDAP_SSL_KEYFILE_INVALID_FORMAT

    These reason codes can be returned in the ldap_ssl_client_init() API
    reason code field, by ldap_get_option() with LDAP_OPT_EXTERROR to get a
    more detailed error when an SSL error occurs, and by messages from the
    LDAP server.
*/


#define LDAP_SSL_OK                           0   /* Successful Completion  */
#define LDAP_SSL_INITIALIZE_OK                0   /* Successful Completion  */
#define LDAP_SSL_KEYFILE_IO_ERROR             1   /* Keyring I/O error      */
#define LDAP_SSL_KEYFILE_OPEN_FAILED          2   /* Keyring open error     */
#define LDAP_SSL_KEYFILE_BAD_FORMAT           3   /* Keyring format bad     */
#define LDAP_SSL_KEYFILE_BAD_PASSWORD         4   /* Keyring PW is incorrect*/
#define LDAP_SSL_KEYFILE_BAD_MALLOC           5   /* OBSOLETE               */
#define LDAP_SSL_KEYFILE_NOTHING_TO_WRITE     6   /* OBSOLETE               */
#define LDAP_SSL_KEYFILE_WRITE_FAILED         7   /* OBSOLETE               */
#define LDAP_SSL_KEYFILE_NOT_FOUND            8   /* OBSOLETE               */
#define LDAP_SSL_KEYFILE_BAD_DNAME            9   /* Distinguished name bad */
#define LDAP_SSL_KEYFILE_BAD_KEY             10   /* OBSOLETE               */
```

```
#define LDAP_SSL_KEYFILE_KEY_EXISTS          11   /* OBSOLETE              */
#define LDAP_SSL_KEYFILE_BAD_LABEL           12   /* Keyfile label is not
                                                     valid, OR certificate
                                                     is not trusted        */
#define LDAP_SSL_KEYFILE_DUPLICATE_NAME      13   /* Key database contains
                                                     multiple certificates
                                                     with the same subject
                                                     name as the
                                                     distinguished name
                                                     specified in the
                                                     connection init data  */
#define LDAP_SSL_KEYFILE_DUPLICATE_KEY       14   /* OBSOLETE              */
#define LDAP_SSL_KEYFILE_DUPLICATE_LABEL     15   /* OBSOLETE              */
#define LDAP_SSL_ERR_INIT_PARM_NOT_VALID    100   /* Initialization Parm is
                                                     not valid             */
#define LDAP_SSL_INIT_HARD_RT               101   /* No keyring file
                                                     or password           */
#define LDAP_SSL_INIT_SEC_TYPE_NOT_VALID    102   /* Security type bad     */
#define LDAP_SSL_INIT_V2_TIMEOUT_NOT_VALID  103   /* V2 timeout value bad  */
#define LDAP_SSL_INIT_V3_TIMEOUT_NOT_VALID  104   /* V3 timeout value bad  */
#define LDAP_SSL_KEYFILE_CERT_EXPIRED       105   /* OBSOLETE              */
#define LDAP_SSL_BAD_FORMAT_OR_INVALID_PASSWORD 106 /* Key database file is
                                                       corrupted           */
#define LDAP_SSL_KEYFILE_NO_CA_CERTIFICATES 109   /* The key database or
                                                     SAF key ring does not
                                                     contain any valid
                                                     certification authority
                                                     certificates.         */
#define LDAP_SSL_NO_KEYFILE_PASSWORD        201   /* Key database PW or stash
                                                     file name not set.    */
#define LDAP_SSL_RSA_TEMP_KEY_PAIR          203   /* Unable to generate
                                                     temporary RSA
                                                     public/private key pair*/
#define LDAP_SSL_KEYFILE_PASSWORD_EXPIRED   204   /* key database password
                                                     is expired.           */
#define LDAP_SSL_CLOSE_FAILED               301   /* Close failed          */
#define LDAP_SSL_CONNECTION_ACTIVE          302   /* Connection has an
                                                     active write          */
#define LDAP_SSL_ERR_BAD_DATE               401   /* Validity time period
                                                     for the certificate
                                                     has expired.          */
#define LDAP_SSL_ERR_LDAP_NOT_AVAILABLE     427   /* Unable to access the
                                                     specified LDAP
                                                     directory, should not
                                                     occur.                */
#define LDAP_SSL_ERR_NO_PRIVATE_KEY         428   /* The specified key did
                                                     not contain a private
                                                     key.                  */
#define LDAP_SSL_ERR_INVALID_V2_HEADER      429   /* SSL V2 header is not
                                                     valid                 */
#define LDAP_SSL_ERR_CERTIFICATE_REVOKED    431   /* Certificate has been
                                                     revoked by the
                                                     certification authority*/
#define LDAP_SSL_ERR_NO_NEGOTIATION         432   /* Session renegotiation
                                                     is not allowed        */
#define LDAP_SSL_ERR_EXPORT_RESTRICTION     433   /* Key exceeds allowable
                                                     export size           */
#define LDAP_SSL_ERR_INCOMPATIBLE_KEY       434   /* Certificate key is not
                                                     compatible with the
                                                     negotiated cipher
                                                     suite                 */
#define LDAP_SSL_ERR_UNKNOWN_CA             435   /* A certification
                                                     authority certificate
                                                     is missing            */
#define LDAP_SSL_ERR_BAD_CRL                436   /* Certificate revocation
                                                     list cannot be
                                                     processed.            */
```

```
#define LDAP_SSL_ERR_CONNECTION_CLOSED      437   /* A close notification
                                                     alert has been sent
                                                     for the connection     */
#define LDAP_SSL_ERR_INTERNAL_ERROR_ALERT   438   /* Internal error reported
                                                     by remote partner      */
#define LDAP_SSL_ERR_UNKNOWN_ALERT          439   /* Unknown alert received
                                                     from remote partner    */
#define LDAP_SSL_INVALID_BUFFER_SIZE        501   /* The buffer size is
                                                     negative or zero.      */
#define LDAP_SSL_WOULD_BLOCK                502   /* Used with non-blocking
                                                     I/O.                   */
#define LDAP_SSL_WOULD_BLOCK_READ           503   /* Read would be blocked;
                                                     Used w/ blocking I/O.  */
#define LDAP_SSL_WOULD_BLOCK_WRITE          504   /* Write would be blocked;
                                                     Used w/blocking I/O.   */
#define LDAP_SSL_ERR_RECORD_OVERFLOW        505   /* Record overflow        */
#define LDAP_SSL_ERR_NOT_SSLV3              601   /* Session is not using
                                                     the SSL V3 or TLS V1
                                                     protocol. This should
                                                     not occur.             */
#define LDAP_SSL_MISC_INVALID_ID            602   /* Function identifier is
                                                     not valid              */
#define LDAP_SSL_ATTRIBUTE_INVALID_ID       701   /* Attribute ID is not
                                                     valid.                 */
#define LDAP_SSL_ATTRIBUTE_INVALID_LENGTH   702   /* Attribute length is
                                                     not valid.             */
#define LDAP_SSL_ATTRIBUTE_INVALID_ENUMERATION 703 /* Attribute enumeration
                                                     value is not valid.    */
#define LDAP_SSL_ATTRIBUTE_INVALID_SID_CACHE  704 /* Setting of session ID
                                                     callback routines
                                                     requires all session
                                                     id routines to be
                                                     specified, should not
                                                     occur.                 */
#define LDAP_SSL_ATTRIBUTE_INVALID_NUMERIC_VALUE 705 /* Attribute value is
                                                        not valid.          */
#define LDAP_SSL_ATTRIBUTE_INVALID_PARAMETER 706  /* Attribute parameter
                                                     value is not valid     */
#define LDAP_SSL_INVALID_HANDLE           10001   /* Environment or SSL
                                                     handle not valid.      */
#define LDAP_SSL_INTERNAL_ERROR           10003   /* Internal SSL error.    */
#define LDAP_SSL_CERTIFICATE_NOT_AVAILABLE 10007  /* Certificate not received
                                                     from partner.          */
#define LDAP_SSL_CERT_VALIDATION          10008   /* Certificate validation
                                                     error                  */
#define LDAP_SSL_ERR_CRYPTO               10009   /* Error processing
                                                     cryptography           */
#define LDAP_SSL_ERR_ASN                  10010   /* Error validating ASN.1
                                                     fields in certificate. */
#define LDAP_SSL_ERR_LDAP                 10011   /* Error connecting to LDAP
                                                     server.  This should
                                                     not occur              */
#define LDAP_SSL_KEYFILE_INVALID_FORMAT   10103   /* The database is not a
                                                     key database           */
#define LDAP_SSL_ERROR_NO_CIPHERS            -1   /* No ciphers matched
                                                     the server and clients
                                                     lists of acceptable
                                                     ciphers                */
#define LDAP_SSL_ERROR_NO_CERTIFICATE        -2   /* No client certificate
                                                     is to be used.         */
#define LDAP_SSL_ERROR_BAD_CERTIFICATE       -4   /* OBSOLETE               */
#define LDAP_SSL_ERROR_UNSUPPORTED_CERTIFICATE_TYPE -6  /* The certificate
                                                           type is not
                                                           supported by
                                                           System SSL       */
#define LDAP_SSL_ERROR_IO                   -10   /* I/O error communicating
                                                     with peer application  */
```

```
#define LDAP_SSL_ERROR_BAD_MESSAGE         -11   /* Incorrectly-formatted
                                                    message received from
                                                    peer application      */
#define LDAP_SSL_ERROR_BAD_MAC             -12   /* Message verification
                                                    failed                */
#define LDAP_SSL_ERROR_UNSUPPORTED         -13   /* SSL protocol or
                                                    certificate type is
                                                    not supported         */
#define LDAP_SSL_ERROR_BAD_CERT_SIG        -14   /* Certificate signature
                                                    is not correct for a
                                                    certificate received
                                                    from the peer         */
#define LDAP_SSL_ERROR_BAD_CERT            -15   /* Certificate is not
                                                    valid                 */
#define LDAP_SSL_ERROR_BAD_PEER            -16   /* Peer application has
                                                    violated the SSL
                                                    protocol              */
#define LDAP_SSL_ERROR_PERMISSION_DENIED   -17   /* Not authorized to access
                                                    key database or keyring*/
#define LDAP_SSL_ERROR_SELF_SIGNED         -18   /* A self-signed certificate
                                                    cannot be validated   */
#define LDAP_SSL_ERROR_BAD_MALLOC          -20   /* Insufficient storage is
                                                    available             */
#define LDAP_SSL_ERROR_BAD_STATE           -21   /* The environment or
                                                    connection is not in
                                                    the open state        */
#define LDAP_SSL_ERROR_SOCKET_CLOSED       -22   /* Socket connection closed
                                                    by peer application   */
#define LDAP_SSL_ERROR_LDAP_SSL_INITIALIZATION_FAILED   -23  /* OBSOLETE    */
#define LDAP_SSL_ERROR_HANDLE_CREATION_FAILED           -24  /* OBSOLETE    */
#define LDAP_SSL_SOC_BAD_V2_CIPHER         -40   /* V2 cipher is not valid */
#define LDAP_SSL_SOC_BAD_V3_CIPHER         -41   /* V3 cipher is not valid */
#define LDAP_SSL_SOC_BAD_SEC_TYPE          -42   /* OBSOLETE               */
#define LDAP_SSL_SOC_NO_READ_FUNCTION      -43   /* OBSOLETE               */
#define LDAP_SSL_SOC_NO_WRITE_FUNCTION     -44   /* OBSOLETE               */
#define LDAP_SSL_SOC_BAD_SEC_TYPE_COMBINATION -102 /* OBSOLETE             */
#define LDAP_SSL_ERROR_UNKNOWN_ERROR       -99   /* Unrecognized error     */
#define LDAP_SSL_DLL_LOAD_FAILED         -1000   /* Failed loading SSL's
                                                        DLL                */
#define LDAP_SSL_ROUTINE_MISSING         -1001   /* Failed to locate an
                                                        SSL function       */
#define LDAP_SSL_LDAP_LOCKINIT_FAILED    -1002   /* Failed to initialize an
                                                        LDAP owned lock    */



#endif /* _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 $(OBJS2) $(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 example 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 <unistd.h>
#include <ldap.h>
#include <locale.h>

#define _XOPEN_SOURCE_EXTENDED 1
#include <signal.h>                          /* sigignore() */

#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 = "127.0.0.1";
static int  ldapport = LDAP_PORT;
static int  not = FALSE;
static int  verbose = FALSE;
static int  contoper = FALSE;
static int  follow_referrals = LDAP_OPT_ON;
static int  deref = LDAP_DEREF_NEVER;
static int ldapversion = LDAP_VERSION3;
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 );


#ifdef __cplusplus
extern "C" {
#endif
int rebindproc( LDAP *ld, char **dnp, char **pwp, int *methodp, int freeit );
int  krb5_rebindproc( LDAP *ld, char **dnp, char **pwp, int *methodp, int
                              freeit );
#ifdef __cplusplus
}
#endif


main( int argc, char **argv )
{
    char    *optpattern = "nvRMZc?h:V:p:D:w:d:f:K:P:N:S:m:U:g:";
    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;
    char    *debugLevel = NULL;
    int     failureReasonCode ;
    extern  char *optarg;
    extern  int  optind;
    char    *mechanism = NULL;
    int     sasl_bind = FALSE;
    struct  berval **servercred = NULL;
    int     host_named = FALSE;
    int     krb5_bind = FALSE;
    char    *username = NULL;
    char    *realmname = NULL;
    struct  berval cred;
    int     mand_auth_bind = FALSE;
    LDAPControl *userControl = NULL;
    LDAPControl *realmControl = NULL;
    LDAPControl *md5_Controls[3];


    setlocale( LC_ALL, "" );

    if ( prog = strrchr( argv[0], '/' ) ) {  /* Strip off any path info
                                              * on program name
                                              */
        ++prog;
    }
    else {
        prog = argv[0];
    }

    sigignore(SIGPIPE);                 /* Ignore possible PIPE errors
                                           generated by the sockets */

    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 );
            host_named = TRUE;
            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':
            if ( (debugLevel = strdup( optarg )) == NULL ) {
                fprintf( stderr, "Out of memory error encountered.\n");
                exit( 1 );
            }
            else {
                rc = ldap_set_option_np( NULL, LDAP_OPT_DEBUG_STRING, debugLevel );
                free( debugLevel );
                if ( rc == LDAP_NO_MEMORY ) {
                    fprintf( stderr, "Out of memory error encountered.\n");
                    exit( 1 );
                }
                else if ( rc != LDAP_SUCCESS ) {
                    fprintf( stderr, "The debug value is not valid.\n");
                    exit( 1 );
                }
            }
            break;
        case 'p':
            ldapport = atoi( optarg );
            port = TRUE;
            break;
        case 'S':   /* use Sasl Bind functions */
        case 'm':
            if ( strncasecmp( optarg, "external", 8 ) == 0 ) {
                sasl_bind = TRUE;
                mechanism = LDAP_MECHANISM_EXTERNAL;
            } else if ( strncasecmp( optarg, "GSSAPI", 6 ) == 0 ) {
                sasl_bind = TRUE;
                krb5_bind = TRUE;
                mechanism = LDAP_MECHANISM_GSSAPI;
            } else if ( strncasecmp( optarg, "CRAM-MD5", 8) == 0 ) {
                sasl_bind = TRUE;
                mand_auth_bind = TRUE;
                mechanism = LDAP_MECHANISM_CRAM;
            } else if ( strncasecmp( optarg, "DIGEST-MD5", 10) == 0 ) {
                sasl_bind = TRUE;
                mand_auth_bind = TRUE;
                mechanism = LDAP_MECHANISM_DIGEST;
            } else {
                fprintf( stderr, "supported mechanisms are EXTERNAL, GSSAPI, CRAM-MD5 and DIGEST-MD5\n" );
                usage( prog );
                exit( 1 );
            }
            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 'g':
            realmname = strdup ( optarg );
            break;
        case 'U':
            username = strdup ( optarg );
            break;
        case '?':
        default:
            usage( prog );
            exit( 1 );
        }
    }

    if ( manageDsa && ( ldapversion == LDAP_VERSION2 ) ) {
        fprintf( stderr, "-M option requires version 3.\n" );
        exit(1);
    }

    /*  A ldap_sasl_bind requires a ldapversion of 3. */
    if (sasl_bind && (ldapversion == LDAP_VERSION2)) {
        fprintf( stderr, "-S/-m option requires version 3.\n");
        usage(prog);
        exit( 1 );
    }

    /* DIGEST-MD5 bind requires username. */
    if ( (strcmp(mechanism, LDAP_MECHANISM_DIGEST) == 0)  && (username == NULL) ) {
        fprintf( stderr, "DIGEST-MD5 bind requires -U option.\n");
        usage(prog);
        exit( 1 );
    }


    /*  If the host was not specified, then attempt to get the host that this application
        is running on DNS name.  If we are unable to resolve its name, then use the string
        127.0.0.1 to represent the localhost.
    */

    if (!host_named) {
       char temp[200];
       int worked = gethostname(temp, 200);
       if (worked == 0) {
           ldaphost = strdup( temp );
       }
    }
    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"
                         " reason text: %s\n",
                         rc, failureReasonCode, ldap_err2string(rc) );
                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 );
        ldap_set_option_np( ld, LDAP_OPT_DEREF, deref );
        ldap_set_option_np( ld, LDAP_OPT_REFERRALS, follow_referrals );

        if (krb5_bind) {
            ldap_set_rebind_proc( ld, krb5_rebindproc );
        } else if ( binddn != NULL && mand_auth_bind == FALSE ) {
            ldap_set_rebind_proc( ld, rebindproc );
        }

        if ( ldapversion != LDAP_VERSION2 && sasl_bind == TRUE ) {
            if ( (!strcasecmp(mechanism,LDAP_MECHANISM_CRAM)) ||
                 (!strcasecmp(mechanism,LDAP_MECHANISM_DIGEST)) ) {

                if ( (userControl = (LDAPControl *)malloc(sizeof(LDAPControl) ) ) == NULL) {
                    fprintf(stderr, "Out of memory error encountered.\n");
                    exit( 1 );
                }

                if ( (realmControl = (LDAPControl *)malloc(sizeof(LDAPControl) ) ) == NULL) {
                    fprintf(stderr, "Out of memory error encountered.\n");
                    exit( 1 );
                }

                userControl->ldctl_oid = IBM_CLIENT_MD5_USER_NAME_OID;
                userControl->ldctl_value.bv_len = strlen(username);
                userControl->ldctl_value.bv_val = username;
                userControl->ldctl_iscritical = LDAP_OPT_OFF;

                realmControl->ldctl_oid = IBM_CLIENT_MD5_REALM_NAME_OID;
                realmControl->ldctl_value.bv_len = strlen(realmname);
                realmControl->ldctl_value.bv_val = realmname;
                realmControl->ldctl_iscritical = LDAP_OPT_OFF;

                md5_Controls[0] = userControl;
                md5_Controls[1] = realmControl;
                md5_Controls[2] = NULL;

                cred.bv_len = strlen ( passwd );
                cred.bv_val = strdup ( passwd );

                if ( ldap_sasl_bind_s(ld, binddn, mechanism, &cred, NULL,
                                      (LDAPControl **)&md5_Controls,
                                      servercred) != LDAP_SUCCESS ) {
                    ldap_perror( ld, "ldap_sasl_bind_s" );
                    exit( 1 );
                }
            } else {
                /* Kerberos and EXTERNAL */
                if ( ldap_sasl_bind_s(ld, NULL, mechanism, NULL, NULL, NULL,
                                      servercred) != LDAP_SUCCESS ) {
                    ldap_perror( ld, "ldap_sasl_bind_s" );
                    exit( 1 );
                }
            }
        } else 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 );
    }

    if (userControl != NULL) {
        free(userControl);
    }

    if (realmControl != NULL) {
        free(realmControl);
    }

    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         distinguished name of entry to delete\n" );
    fprintf( stderr, "    entryfile file containing DNs to delete\n" );
    fprintf( stderr, "              on consecutive lines\n" );
    fprintf( stderr, "options:\n" );
    fprintf( stderr, "    -?            print this text\n" );
    fprintf( stderr, "    -V version    select LDAP protocol version"
                                   " (2 or 3; default is 3)\n");
    fprintf( stderr, "    -S mechanism  select SASL bind mechanism"
           " (supported mechanisms are EXTERNAL, GSSAPI, CRAM-MD5 and DIGEST-MD5)\n");
    fprintf( stderr, "    -m mechanism  select SASL bind mechanism"
             " (supported mechanisms are EXTERNAL, GSSAPI, CRAM-MD5, and DIGEST-MD5)\n");
    fprintf( stderr, "    -c            continue even if error encountered\n");
    fprintf( stderr, "    -n            show what would be done but don't actually"
                                  " delete\n" );
    fprintf( stderr, "    -v            run in verbose mode (diagnostics to"
                                  " standard output)\n" );
    fprintf( stderr, "    -R            do not automatically follow referrals\n" );
    fprintf( stderr, "    -M            Treat referral objects as normal entries."
                                  " (requires -V 3)\n" );
    fprintf( stderr, "    -d level      set LDAP debugging level to 'level'\n" );
    fprintf( stderr, "    -f file       perform sequence of deletes listed"
                                  " in 'file'\n" );
    fprintf( stderr, "    -D binddn     bind dn\n" );
    fprintf( stderr, "    -w passwd     bind passwd\n" );
    fprintf( stderr, "    -h host       ldap server\n" );
    fprintf( stderr, "    -p port       port on ldap server\n" );
    fprintf( stderr, "    -Z            use a secure ldap connection for the"
                                   " operation\n");
    fprintf( stderr, "    -K keyfile    file to use for keys/certificates\n");
    fprintf( stderr, "    -P key_pw     keyfile password\n");
    fprintf( stderr, "    -N key_dn     Certificate Name in keyfile\n");
    fprintf( stderr, "    -g realm      Mandatory Authentication realm\n");
    fprintf( stderr, "    -U username   Mandatory Authentication username (uid) \n");
    fprintf( stderr, "\nRefer to \"z/OS Security Server LDAP Client Programming"
             " Guide\", Document Number: SC24-5924, for complete documentation\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 {
        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 );
}



int  krb5_rebindproc( LDAP *ld, char **dnp, char **pwp, int *methodp,
                        int freeit )
{

    *methodp = LDAP_AUTH_SASL_30;
    *dnp = NULL;
    *pwp = NULL;
    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 8 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* 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 <unistd.h>
#include <locale.h>

#define _XOPEN_SOURCE_EXTENDED 1
#include <signal.h>                          /* sigignore() */

#ifndef TRUE
    #define TRUE 1
#endif
```

```
#ifndef FALSE
    #define FALSE 0
#endif

#define DEFSEP "="


#ifdef __cplusplus
extern "C" {
#endif
int  rebindproc( LDAP *ld, char **dnp, char **pwp, int *methodp, int
                              freeit );
int  krb5_rebindproc( LDAP *ld, char **dnp, char **pwp, int *methodp, int
                             freeit );
#ifdef __cplusplus
}
#endif

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 = "127.0.0.1";
static int  ldapport = LDAP_PORT;
static char *sep = DEFSEP;
static int  verbose, not, allow_binary, print_local, vals2tmp, ldif;
static int  ldapversion = LDAP_VERSION3;

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};


main( int argc, char **argv ) {
    char    *optpattern = "?ZnvtRMABCLD:V:s:f:h:b:d:p:F:a:w:l:z:S:K:P:N:m:U:g:";
    char    *infile, *filtpattern, **attrs, line[ BUFSIZ ];
    FILE    * fp;
    int     rc, i, first, scope, deref, attrsonly, port = 0;
    int     timelimit, sizelimit;
    int     follow_referrals;
    LDAP    * ld;
    extern  char *optarg;
    extern  int  optind;
    char    * debugLevel = NULL;
    int     ssl = FALSE;
    char    *keyfile = NULL, *keyfile_pw = NULL, *keyfile_dn = NULL;
    int     failureReasonCode;
    FILE    * cf_fd;
    char    *mechanism = NULL;
    int     sasl_bind = FALSE;
    int     host_named = FALSE;
    struct  berval **servercred = NULL;
    int     manageDsa = FALSE;
    int     krb5_bind = FALSE;
    struct  berval cred;
    char    *realmname = NULL;
    char    *username = NULL;
    int     mand_auth_bind = FALSE;
    LDAPControl *userControl = NULL;
    LDAPControl *realmControl = NULL;
    LDAPControl *md5_Controls[3];
    if (prog = strrchr(argv[0], '/'))   /* Strip off any path info
                                         * on program name
                                         */
        ++prog;
    else
        prog = argv[0];

    sigignore(SIGPIPE);                 /* Ignore possible PIPE errors
                                           generated by the sockets */

    setlocale(LC_ALL, "");

    infile = NULL;
    deref = verbose = allow_binary = print_local = 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 version level supplied.\n");
                fprintf( stderr, "Supported values for the -V parameter"
                         " are 2 and 3\n");
                usage( prog );
                exit( 1 );
            }
            break;
        case 'S':   /* use Sasl Bind functions */
        case 'm':
            if ( strncasecmp( optarg, "external", 8 ) == 0 ) {
                sasl_bind = TRUE;
                mechanism = LDAP_MECHANISM_EXTERNAL;
            }
            else if ( strncasecmp( optarg, "GSSAPI", 6 ) == 0 ) {
                krb5_bind = TRUE;
                sasl_bind = TRUE;
                mechanism = LDAP_MECHANISM_GSSAPI;
            }
            else if ( strncasecmp( optarg, "CRAM-MD5", 8) == 0 ) {
                mand_auth_bind = TRUE;
                sasl_bind = TRUE;
                mechanism = LDAP_MECHANISM_CRAM;
            }
            else if ( strncasecmp( optarg, "DIGEST-MD5", 10) == 0 ) {
                mand_auth_bind = TRUE;
                sasl_bind = TRUE;
                mechanism = LDAP_MECHANISM_DIGEST;
            }
            else {
                fprintf( stderr, "supported mechanisms are EXTERNAL, GSSAPI, CRAM-MD5, and DIGEST-MD5\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':
            if ( (debugLevel = strdup( optarg )) == NULL ) {
                fprintf( stderr, "Out of memory error encountered.\n");
                exit( 1 );
            }
            else {
                rc = ldap_set_option_np( NULL, LDAP_OPT_DEBUG_STRING, debugLevel );
                free( debugLevel );
                if ( rc == LDAP_NO_MEMORY ) {
                    fprintf( stderr, "Out of memory error encountered.\n");
                    exit( 1 );
                }
                else if ( rc != LDAP_SUCCESS ) {
                    fprintf( stderr, "The debug value is not valid.\n");
                    exit( 1 );
                }
            }
            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;
            allow_binary = TRUE; /* always allow binary when outputting LDIF */
            break;
        case 'B':   /* allow binary values to be printed */
            allow_binary = TRUE;
            break;
        case 'C':   /* allow multi-byte UTF-8 characters to be printed */
            print_local = 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 );
            host_named = TRUE;
            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;
        case 'U':
            username = strdup( optarg );
            break;
        case 'g':
            realmname = strdup( optarg );
            break;
        case '?':
            usage( prog );
            exit( 0 );
        default:
            usage( prog );
            exit( 1 );
        }
    }

    if ( manageDsa && ( ldapversion == LDAP_VERSION2 ) ) {
        fprintf( stderr, "-M option requires version 3. -M ignored.\n");
        manageDsa = FALSE;
    }


    /*  A ldap_sasl_bind requires a ldapversion of 3. */
    if (sasl_bind && (ldapversion == LDAP_VERSION2)) {
        fprintf( stderr, "-S/-m option requires version 3.\n");
        usage(prog);
        exit( 1 );
    }


    /* DIGEST-MD5 bind requires username. */
    if ( (strcmp(mechanism, LDAP_MECHANISM_DIGEST) == 0)  && (username == NULL) ) {
        fprintf( stderr, "DIGEST-MD5 bind requires -U option.\n");
        usage(prog);
        exit( 1 );
    }


    /*  If the host was not specified, then attempt to get the host that this application
        is running on DNS name.  If we are unable to resolve its name, then use the string
        127.0.0.1 to represent the localhost.
    */

    if (!host_named) {
        char temp[200];
        int worked = gethostname(temp, 200);
        if (worked == 0) {
            ldaphost = strdup( temp );
        }

    }


    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 );
        exit( 1 );
    }
    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"
                     " reason text: %s\n",
                     rc, failureReasonCode, ldap_err2string(rc) );
            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 ) {
            fprintf( stderr, "ldap_init failed; LDAP Handle is NULL.\n");
            exit( 1 );
        }
    }

    ldap_set_option_np( ld, LDAP_OPT_PROTOCOL_VERSION, ldapversion );
    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 (krb5_bind) {
        ldap_set_rebind_proc( ld, krb5_rebindproc );
    }
    else if ( binddn != NULL && mand_auth_bind == FALSE) {
        ldap_set_rebind_proc( ld, rebindproc );
    }

    if ( ldapversion != LDAP_VERSION2 && sasl_bind == TRUE ) {
        if ( (!strcasecmp(mechanism,LDAP_MECHANISM_CRAM)) ||
             (!strcasecmp(mechanism,LDAP_MECHANISM_DIGEST)) ) {

            if ( (userControl = (LDAPControl *)malloc(sizeof(LDAPControl) ) ) == NULL) {
                fprintf(stderr, "Out of memory error encountered.\n");
                exit( 1 );
            }

            if ( (realmControl = (LDAPControl *)malloc(sizeof(LDAPControl) ) ) == NULL) {
                fprintf(stderr, "Out of memory error encountered.\n");
                exit( 1 );
            }

            userControl->ldctl_oid = IBM_CLIENT_MD5_USER_NAME_OID;
            userControl->ldctl_value.bv_len = strlen(username);
            userControl->ldctl_value.bv_val = username;
            userControl->ldctl_iscritical = LDAP_OPT_OFF;

            realmControl->ldctl_oid = IBM_CLIENT_MD5_REALM_NAME_OID;
            realmControl->ldctl_value.bv_len = strlen(realmname);
            realmControl->ldctl_value.bv_val = realmname;
            realmControl->ldctl_iscritical = LDAP_OPT_OFF;

            md5_Controls[0] = userControl;
            md5_Controls[1] = realmControl;
            md5_Controls[2] = NULL;

            cred.bv_len = strlen ( passwd );
            cred.bv_val = strdup ( passwd );
```

```
                if ( ldap_sasl_bind_s(ld, binddn, mechanism, &cred, NULL,
                                      (LDAPControl **) &md5_Controls,
                                      servercred) != LDAP_SUCCESS ) {
                    ldap_perror( ld, "ldap_sasl_bind_s" );
                    exit( 1 );
                }
            }
            else {
                /* Kerberos and EXTERNAL */
                if ( ldap_sasl_bind_s(ld, NULL, mechanism, NULL, NULL, NULL,
                                      servercred) != LDAP_SUCCESS ) {
                    ldap_perror( ld, "ldap_sasl_bind_s" );
                    exit( 1 );
                }
            }
        }
        else 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 );
            }
        }

        if ( manageDsa ) {
            ldap_set_option_np( ld, LDAP_OPT_SERVER_CONTROLS, M_controls );
        }

    } /* ! 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 );
    }

    if (userControl != NULL) {
        free(userControl);
    }

    if (realmControl != NULL) {
        free(realmControl);
    }

    exit( rc );
}

static void usage( char *s ) {
    fprintf( stderr, "usage: %s [options] filter [attributes...]\nwhere:\n",
             s );
    fprintf( stderr, "    filter    RFC-1558 compliant LDAP search filter\n" );
    fprintf( stderr, "    attributes    whitespace-separated list of"
             " attributes to retrieve\n" );
    fprintf( stderr, "        (if no attribute list is given, all are"
             " retrieved)\n" );
    fprintf( stderr, "options:\n" );
    fprintf( stderr, "    -?            print this text\n" );
    fprintf( stderr, "    -V version    select LDAP protocol version"
             " (2 or 3; default is 3)\n");
    fprintf( stderr, "    -S mechanism  select SASL bind mechanism"
             " (supported mechanisms are EXTERNAL, GSSAPI, CRAM-MD5, and DIGEST-MD5)\n");
    fprintf( stderr, "    -m mechanism  select SASL bind mechanism"
             " (supported mechanisms are EXTERNAL, GSSAPI, CRAM-MD5, and DIGEST-MD5)\n");
    fprintf( stderr, "    -n            show what would be done but don't actually"
             " search\n" );
    fprintf( stderr, "    -v            run in verbose mode (diagnostics to standard"
             " output)\n" );
    fprintf( stderr, "    -t            write values to files in /tmp\n" );
    fprintf( stderr, "    -A            retrieve attribute names only (no values)\n" );
    fprintf( stderr, "    -B            do not suppress printing of non-printable"
             " values (printed in wire format)\n" );
    fprintf( stderr, "    -C            do not suppress printing of printable non-ascii"
             " values (printed in local codepage)\n" );
    fprintf( stderr, "    -L            print entries in LDIF format"
             " (-B is implied)\n" );
    fprintf( stderr, "    -R            do not automatically follow referrals\n" );
    fprintf( stderr, "    -M            Manage referral objects as normal entries."
             " (requires -V 3)\n" );
    fprintf( stderr, "    -d level      set LDAP debugging level to 'level'\n" );
    fprintf( stderr, "    -F sep        print `sep' instead of `=' between"
             " attribute names and values\n" );
    fprintf( stderr, "    -f file       perform sequence of searches listed in"
             " 'file'. ('-' implies stdin)\n" );
    fprintf( stderr, "    -b basedn     base dn for search. LDAP_BASEDN in"
             " environment is default\n" );
    fprintf( stderr, "    -s scope      one of base, one, or sub"
             " (search scope)\n" );
    fprintf( stderr, "    -a deref      one of never, always, search, or"
             " find (alias dereferencing)\n" );
    fprintf( stderr, "    -l time lim   time limit (in seconds) for search\n" );
    fprintf( stderr, "    -z size lim   size limit (in entries) for search\n" );
    fprintf( stderr, "    -D binddn     bind dn\n" );
    fprintf( stderr, "    -w passwd     bind passwd\n" );
    fprintf( stderr, "    -h host       ldap server\n" );
    fprintf( stderr, "    -p port       port on ldap server\n" );
    fprintf( stderr, "    -Z            use a secure ldap connection for search\n");
    fprintf( stderr, "    -K keyfile    file to use for keys/certificates\n");
    fprintf( stderr, "    -P key_pw     keyfile password\n");
    fprintf( stderr, "    -N key_dn     Certificate Name in keyfile\n");
    fprintf( stderr, "    -g realm      Mandatory Authentication realm\n");
    fprintf( stderr, "    -U username   Mandatory Authentication username (uid) \n");
    fprintf( stderr, "\nRefer to \"z/OS Security Server LDAP Client Programming"
             " Guide\", Document Number: SC24-5924, for complete documentation\n");
}
```

```
static int  dosearch( LDAP *ld, char *base, int scope, char **attrs,
                      int attrsonly, char *filtpatt, char *value ) {
    char    filterOnStack[ BUFSIZ ], **val, *filter;
    int rc, first, matches, freeFilter;
    int references;
    char    **referrals = NULL;
    int errcode;
    char    *matched, *errmsg;
    LDAPMessage     * res, *e;
    int     msgidp;
    int     filterLength;


    if (value) {
        filterLength = strlen(filtpatt) + strlen(value);
    }
    else {
        filterLength = strlen(filtpatt);
    }

    if (filterLength < BUFSIZ ) {
        filter = filterOnStack;
        freeFilter = 0;
    }
    else {
        if ((filter= (char *) malloc(filterLength+1)) == NULL) {
            fprintf( stderr, "Unable to allocate storage for filter (%d)\n", filterLength);
            return(LDAP_NO_MEMORY);
        }
        freeFilter = 1;
    }



    if ( value ) {
        sprintf( filter, filtpatt, value );
    }
    else {
        if (filterLength > BUFSIZ) {
            strncpy ( filter, filtpatt, filterLength );
        }
        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 );
        }
    }

    if (freeFilter) {
        free(filter);
    }

    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)) != NULL)) {
            for ( i = 0; bvals[i] != NULL; i++) {
                if ( vals2tmp ) {
                    sprintf( tmpfname, "/tmp/ldapsearch-%s-XXXXXX", a );
                    tmpfp = NULL;

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

                    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 {
                        int str_value_len = strlen(str_value);
                        printable = TRUE;
                        /* if print_local==TRUE, don't perform the following
                         * length check because a string containing multi-byte
                         * UTF-8 characters may not match the length of the same
                         * string as represented in the local codepage.  Defer
                         * all "printable" checking to 'isprint()/isspace()' if
                         * print_local==TRUE.
                         */
                        if ( print_local || (str_value_len == value_len) ) {
                            for ( j = 0; j < str_value_len; j++) {
                                if ( !isprint( str_value[ j ] ) &&
                                     !isspace( str_value[ j ] ) ) {
                                    printable = FALSE;
                                    break;
                                }
                            }
                        }
                        else {
                            printable = FALSE;
                        }

                        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 );
        }
        else {
            /* ldap_get_values_len returned NULL.  This means that either
               an error occurred or there were no values.  Check the
               ldap errno.
            */
            if (ldap_get_errno(ld) != LDAP_SUCCESS) {
                fprintf(stderr,
                        "ldap_search: ldap_get_values_len failed for attribute=%s. Return Code=%d, %s\n",
                        a, ldap_get_errno(ld), ldap_err2string(ldap_get_errno(ld)));
                exit(1);
            }
        }

        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 );
}


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 );
}


int  krb5_rebindproc( LDAP *ld, char **dnp, char **pwp, int *methodp,
                             int freeit ) {

    *methodp = LDAP_AUTH_SASL_30;
    *dnp = NULL;
    *pwp = NULL;
    return( LDAP_SUCCESS );
}
```

# Appendix D. Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS™ enable users to:

- Use assistive technologies such as screen-readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

## Using assistive technologies

Assistive technology products, such as screen-readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using it to access z/OS interfaces.

## Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to *z/OS: TSO/E Primer*, *z/OS: TSO/E User's Guide*, and *z/OS: ISPF User's Guide Volume I* for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

# 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

## Programming interface information

This *z/OS: Security Server LDAP Client Programming* book primarily documents intended Programming Interfaces that allow the customer to write programs to obtain services of z/OS LDAP.

This *z/OS: Security Server LDAP Client Programming* book also documents information that is not intended to be used as Programming Interfaces of z/OS LDAP. This information is identified where it occurs with an introductory statement to a chapter.

## Trademarks

| | | | |
|---|---|---|---|
| IBM | IBMLink | OS/2 | Resource Link |
| AIX/6000 | Language Environment | OS/390 | z/OS |
| BookManager | Library Reader | RACF | |

# Bibliography

This bibliography provides a list of publications that are useful when using the LDAP programming interface:

- *z/OS: Security Server LDAP Server Administration and Use*, SC24-5923
- *z/OS: System Secure Sockets Layer Programming*, SC24-5901
- *z/OS: DCE Application Development Guide: Directory Services*, SC24-5906
- *z/OS: UNIX System Services Command Reference*, SA22-7802
- *z/OS: Communications Server: IP Configuration Guide*, SC31-8775
- *z/OS: Language Environment Customization*, SA22-7564
- *z/OS: C/C++ Programming Guide*, SC09-4765
- *z/OS: Security Server Network Authentication Service Administration*, SC24-5926
- *z/OS: Collection*, SK3T-4269
- *z/OS: Information Roadmap*, SA22-7500

# 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 Dictionary of Computing*, available from:

`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.

**DNS.** Domain Name System.

**Domain Name System (DNS).** In the Internet suite of protocols, the distributed database system used to map domain names to IP addresses.

## E

**environment variable.** A variable included in the current software environment that is available to any called program that requests it.

**extended operations.** A generic operation that extends the LDAP protocol. The operation contains an object identifier that uniquely identifies the intended operation. The extended operation allows additional operations to be defined for services not available elsewhere in the LDAP V3 protocol.

## G

**Generic Security Service (GSS) API .** An application programming interface enabling application programs that do not implement remote procedure calls (RPCs) to have security services provided by a server in a Distributed Computing Environment (DCE). The GSS API provides security services to callers through a generic method that functions independently of underlying cryptography mechanisms or communication protocols and can thus be used in many different environments. The GSS API became available as part of the Open Software Foundation's (OSF's) Release 1.1 of DCE.

**GSS API.** Generic Security Service API.

## K

**Kerberos.** The security system of the Massachusetts Institute of Technology's (MIT's) Project Athena. It uses symmetric key cryptography to provide security services to users in a network.

## 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 security protocol that provides communication privacy over the Internet. The protocol allows client/server applications to ommunicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

**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 authentication information outside the client and server.

**SSL.** Secure Sockets Layer.

## T

**thread.** A single sequential flow of control within a process.

**TLS.** Transport Layer Security.

**Transport Layer Security.** A security protocol that provides communication privacy over the Internet. The protocol allows client/server applications to ommunicate in a way that is designed to prevent eavesdropping, tampering, or message forgery. TLS is based upon SSL Version 3.0.

## 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.

# Index

## A

abandoning LDAP operation 28
accessibility 183
accessing RACF information 27
adding entries 10, 30, 121
aliases, following 62
approximate filter 88, 135
asynchronous LDAP operation 17
attribute values
  comparing 12
  counting 54
  retrieving 54
attributes
  counting 48
  LDAP 2
  stepping through 48
  type 2
authentication
  certificate 1
  general 34
  Kerberos 1
  methods 1
  SASL 34
  simple 1, 34

## B

Basic Encoding Rules (BER) 2
batch jobs
  using to run, link, compile 8
BER (Basic Encoding Rules) 2
bibliography 187
bind mechanism 33
binding to Directory Service 9
binding to server 32
binding with SASL GSS API 1
books
  related 187
breaking down LDAP URL 112

## C

C programming language
  utility routines 2
C/C++ programming language
  for SOCKS server 5
  LDAP DLL 3
caching
  client-side search results 17
  search results 70
call-back function 32
cancelling LDAP operation 28
certificate authentication 1
certificates 110
changing entry name 82
changing LDAP entries 76
changing RDN 12, 82
character string, deallocating 73
chasing referrals 61
checking for LDAP URL 112
ciphers 109
ciphers, supported 64
classes, LDAP SPI 17
client and server authentication 108
client API, LDAP 3
client controls 24, 25, 66
client-side caching 17
CNAME alias record 104
command-line utilities
  ldapadd 121
  ldapdelete 118
  ldapmodify 121
  ldapmodrdn 131
  ldapsearch 135
  using 116
comparing LDAP entries 37
compiling program 7, 8
configuration file, socks.conf 5
context LDAP handle 67
continuation references, retrieving 50
controls
  client 25
  LDAP 24
  session 24
conventions in this document ix
counting attributes 48
counting continuation reference 50
counting LDAP entries 50
counting LDAP handles 54
counting LDAP values 54
CRAM-MD5 authentication 1
creating client side cache 70
creating SSL connection 107

## D

data model
  LDAP 2
datasets
  z/OS 2
de-initialize LDAP API 10
deallocating
  array of LDAP values 54
  character strings 73
  LDAP handle 32
  LDAP URL description 112
  LDAP values 54
  memory 85
  storage 6, 73
  structures 76
debug levels 15, 16, 64
debug trace 62, 64
definitions of terms 189
deleting LDAP entries 11, 39, 161

deprecated APIs
  ldap_bind 32
  ldap_bind_s 32
  ldap_modrdn 82
  ldap_modrdn_s 82
  ldap_open 57
  ldap_perror 42
  ldap_result2error 42
  ldap_ssl_start 107
  listing of 19
describing error message 42
DIGEST-MD5 authentication 1
directory
  access protocol (LDAP) 1
  entry
    naming 2
Directory Service
  extracting information from using LDAP 1
disability 183
distinguished name (DN)
  getting from LDAP entry 53
  parsing 53
  specifying with LDAP 3
DLL (dynamic link library)
  C/C++ 3
DNS (Domain Name Service) 41, 92
document organization ix
domain
  eNetwork 41
Domain Name Service (DNS) 41, 92
dynamic link library (DLL) 3

## E

eNetwork domain
  managing 41
entries
  deleting using example program 161
  LDAP 2
  searching using example program 169
environmental variables
  PATH, setting 115
  session settings 60
error code 67
error code, returning 42
error handling
  LDAP 3, 5, 14
error message, describing 42
errors
  printing indication of 42
  retrieving 42
establishing call-back function 32
example Makefile 159
example programs
  deleting entries 161
  searching entries 169
extended error code 68
extended operations 46
extracting information 1
extracting information from results 79

## F

file, configuration 5
filter
  using for search 135
finding
  LDAP server 92
first LDAP entry, getting 50
following aliases 62
freeing
  character strings 73
  LDAP handle 32
  LDAP URL description 112
  LDAP values 54
  list of servers 92
  memory 85
  storage 6, 73
  structures 76
freeing cache 70
function
  call-back 32

## G

general authentication 34
getting
  error codes 42
  first LDAP entry 50
  LDAP attribute values 54
  LDAP DNs 53
  LDAP handles to attribute values 54
  next attribute type name 48
  next LDAP entry 50
  option 57
GLDCLDAP 3
GLDCMMN 3
GLDDHUTD 3
GLDSCKS 3
global cache 70
glossary of terms 189
gskkyman utility 110

## H

handling errors 3, 5
header files
  lber.h 141
  ldap.h 142
  ldapssl.h 154
host name 67

## I

IBM JNDI service provider 18
ibm-saslBindDigestRealmName 26
ibm-saslBindDigestUserName 25
ibm-serverHandledSearchRequest 25
inactivity timer 65
initialization functions 3
Initialization functions 4
initializing context 57

initializing SSL 107
interface
  programming interface information 186
  programming, LDAP 1
interpreting results
  LDAP 5, 13, 42

## J

Java Naming and Directory Interface (JNDI). Se 17
JNDI (Java Naming and Directory Interface) 17

## K

Kerberos
  authentication 1
  credentials 66
key database 110
keyboard 183

## L

lber.h header file 141
LDAP
  adding entry 10
  API functions 3
  asynchronous operation 17
  changing RDN 12
  client for Java 17
  defining 1
  deleting entries 11
  error handling 14
  example programs 161
  header files
    lber.h 141
    ldap.h 142
    ldapssl.h 154
  interface 50
  interface routines 92
    abandoning operation 28
    adding entry 30
    binding to server 32
    caching search results 70
    checking for LDAP URL 112
    comparing LDAP entries 37
    controls 24
    counting attributes 48
    counting continuation reference 50
    counting LDAP entries 50
    counting LDAP values 54
    counting values 54
    creating SSL connection 107
    deallocating array of values 54
    deallocating character strings 73
    deallocating LDAP URL description 112
    deallocating memory 85
    deallocating storage 73
    deallocating structures 76
    deallocating values 54
    deleting LDAP entries 39
    deprecated routines 19

LDAP *(continued)*
  interface routines *(continued)*
    describing error message 42
    establishing call-back function 32
    extracting information from results 79
    finding LDAP server information 92
    freeing character strings 73
    freeing LDAP URL description 112
    freeing LDAP values 54
    freeing storage 73
    freeing structures 76
    getting option 57
    initializing context 57
    initializing SSL 107
    managing eNetwork domain 41
    modifying entries 76
    modifying entry name 82
    modifying RDN 82
    obtaining DNs 53
    obtaining message ID 85
    obtaining message type 85
    opening connection 57
    parsing DNs 53
    parsing URL 112
    performing extended operations 46
    printing error 42
    rebinding 32
    retrieving array of server controls 50
    retrieving attribute values 54
    retrieving error codes 42
    retrieving first entry 50
    retrieving list of continuation references 50
    retrieving next attribute type name 48
    retrieving pointers to attribute values 50
    returning error codes 42
    returning result 85
    searching entries 87
    searching entries with timeout 87
    searching for URL 112
    searching for URL with timeout 112
    session controls 24
    setting option 57
    stepping through attributes 48
    stepping through messages 74
    unbinding 32
    waiting for result 85
  ldapdelete.c 161
  ldapsearch.c 169
  listing all subentries 13
  modifying entry 11
  program structure 9
  program to delete entries 161
  programming 1
  reading an entry's contents 12
  reading attribute values 12
  results, getting 13
  synchronous operation 17
  thread safety 17
  URL 105, 107
    specifying for host 58
  using the API 9

LDAP *(continued)*
  utilities 115
LDAP SPI (service provider interface) 17
ldap_abandon 28
ldap_abandon_ext 28
ldap_add 9, 10, 30
ldap_add_ext 30
ldap_add_ext_s 30
ldap_add_s 11, 30
ldap_bind 9, 32
ldap_bind_s 9, 32
ldap_compare 9, 37
ldap_compare_ext 37
ldap_compare_ext_s 37
ldap_compare_s 12, 37
ldap_control_free 73
ldap_controls_free 73
ldap_count_attributes 48
ldap_count_entries 50
ldap_count_messages 74
ldap_count_references 50
ldap_count_values 54
ldap_count_values_len 54
LDAP_DEBUG 15
ldap_delete 9, 11, 39
ldap_delete_ext 39
ldap_delete_ext_s 39
ldap_delete_s 39
ldap_enetwork_domain_get 41
ldap_enetwork_domain_set 41
ldap_err2string 15, 42
ldap_explode_dn 53
ldap_extended_operation 46
ldap_extended_operation_s 46
ldap_first_attribute 48
ldap_first_entry 50
ldap_first_message 74
ldap_first_reference 50
ldap_free_urldesc 112
ldap_get_dn 53
ldap_get_entry_controls_np 50
ldap_get_errno 14, 42
ldap_get_option 57
ldap_get_values 54
ldap_get_values_len 54
ldap_init 9, 57
ldap_is_ldap_url 112
ldap_memcache_destroy 70
ldap_memcache_flush 70
ldap_memcache_get 70
ldap_memcache_init 70
ldap_memcache_set 70
ldap_memcache_update 70
ldap_memfree 73
ldap_modify 9, 10, 11, 76
ldap_modify_ext 76
ldap_modify_ext_s 76
ldap_modify_s 76
ldap_modrdn 82
ldap_modrdn_s 82
ldap_mods_free 76
ldap_msgfree 85
ldap_msgid 85
ldap_msgtype 85
ldap_next_attribute 48
ldap_next_entry 50
ldap_next_message 74
ldap_next_reference 50
ldap_parse_extended_result 79
ldap_parse_reference_np 50
ldap_parse_result 79
ldap_parse_sasl_bind_result 79
ldap_perror 42
ldap_rename 82
ldap_rename_s 82
ldap_result 5, 14, 85
ldap_result2error 14, 42
ldap_sasl_bind 32
ldap_sasl_bind_s 32
ldap_search 5, 9, 87
ldap_search_ext 87
ldap_search_ext_s 87
ldap_search_s 12, 13, 87
ldap_search_st 87
ldap_server_conf_save 92
ldap_server_free_list 92
ldap_server_locate 92
ldap_set_option 15, 57
ldap_set_option_np 57
ldap_set_rebind_proc 32
ldap_simple_bind 32
ldap_simple_bind_s 32
ldap_ssl_client_init 107
ldap_ssl_init 107
ldap_ssl_start 107
ldap_unbind 10, 57
ldap_unbind_s 10, 57
ldap_url_parse 112
ldap_url_search 112
ldap_url_search_s 112
ldap_url_search_st 112
ldap_value_free 54
ldap_value_free_len 54
ldap.h header file 142
ldapadd utility
  description 121
  running 115
ldapdelete utility
  description 118
  running 115
ldapdelete.c 161
ldapmodify utility
  description 121
  running 115
ldapmodrdn utility
  description 131
  running 115
ldapsearch utility
  description 135
  running 115
ldapsearch.c 169
ldapssl.h header file 154

LDIF mode 123
levels, debug 15
licensed publications x
linking program 7
listing all subentries 13
LookAt messages x
LPALIB 3

## M

Makefile
  example 8, 159
managing eNetwork domain 41
mask
  specifying for debug 15
mechanism,bind 33
message
  stepping through list 74
messages
  using LookAt x
model data
  LDAP 2
modes
  input 123
  LDIF 123
  modify 126
modify mode 126
modifying
  entries 121
  LDAP entries 11
  RDN of entries 131
  schema 129
multiple operations 17

## N

name
  typed 2
nonportable API 57

## O

object class 2
obtaining
  LDAP DNs 53
  LDAP message ID 85
  LDAP message type 85
opening LDAP connection 57
operation utilities, LDAP 115
operations, extended 46
option, setting value of LDAP 57
organization of document ix

## P

parsing
  information from results 79
  LDAP DNs 53
  LDAP URL 112
primitive LDAP operations 3, 5
printing LDAP error 42
processing
  errors 14
  results 5
  URLs 5
programming interface
  LDAP 1
programming interface information 186
protocol
  LDAP 1, 65
pseudo-SRV records 104
publications
  related 187
published LDAP server information
  finding 92
  saving 92

## R

RACF (Resource Access Control Facility) 27
RACF key ring 110
RDN (relative distinguished name)
  changing 12
  examples of LDAP RDNs 3
  modifying 82, 131
  using with LDAP 3
reading attribute values 13
reading entry contents 12
rebinding 32, 35
records
  pseudo-SRV 104
  SRV 101
  TXT 101
referrals 32, 35
relative distinguished name (RDN)
  See RDN 3
removing cached search results 70
removing LDAP entries 39
RESOLVER_CONFIG
  environmental variable 5
Resource Access Control Facility (RACF) 27
Resource Link x
results
  extracting information from 79
  getting with LDAP 13
  processing 5
retriev 54
retrieving
  array of server controls 48
  error codes 42
  first LDAP entry 48
  LDAP attribute values 54
  LDAP entry count 48
  next attribute count type name 48
  next LDAP entry 50
  option 57
returning error codes 42
returning LDAP result 85
root DSE 140
routines
  control 24

routines *(continued)*
  ldap_abandon 28
  ldap_abandon_ext 28
  ldap_add 30
  ldap_add_ext 30
  ldap_add_ext_s 30
  ldap_add_s 30
  ldap_bind 32
  ldap_bind_s 32
  ldap_compare 37
  ldap_compare_ext 37
  ldap_compare_ext_s 37
  ldap_compare_s 37
  ldap_control_free 73
  ldap_controls_free 73
  ldap_count_attributes 48
  ldap_count_entries 50
  ldap_count_messages 74
  ldap_count_references 50
  ldap_count_values 54
  ldap_count_values_len 54
  ldap_delete 39
  ldap_delete_ext 39
  ldap_delete_ext_s 39
  ldap_delete_s 39
  ldap_enetwork_domain_get 41
  ldap_enetwork_domain_set 41
  ldap_err2string 42
  ldap_explode_dn 53
  ldap_extended_operation 46
  ldap_extended_operation_s 46
  ldap_first_attribute 48
  ldap_first_entry 50
  ldap_first_message 74
  ldap_first_reference 50
  ldap_free_urldesc 112
  ldap_get_dn 53
  ldap_get_entry_controls_np 50
  ldap_get_errno 42
  ldap_get_option 57
  ldap_get_values 54
  ldap_get_values_len 54
  ldap_init 57
  ldap_is_ldap_url 112
  ldap_memcache_destrooy 70
  ldap_memcache_flush 70
  ldap_memcache_get 70
  ldap_memcache_init 70
  ldap_memcache_set 70
  ldap_memcache_update 70
  ldap_memfree 73
  ldap_modify 76
  ldap_modify_ext 76
  ldap_modify_ext_s 76
  ldap_modify_s 76
  ldap_modrdn 82
  ldap_modrdn_s 82
  ldap_mods_free 76
  ldap_msgfree 85
  ldap_msgid 85
  ldap_msgtype 85

routines *(continued)*
  ldap_next_attribute 48
  ldap_next_entry 50
  ldap_next_message 74
  ldap_next_reference 50
  ldap_open 57
  ldap_parse_extended_result 79
  ldap_parse_reference_np 50
  ldap_parse_result 79
  ldap_parse_sasl_bind_result 79
  ldap_perror 42
  ldap_rename 82
  ldap_rename_s 82
  ldap_result 85
  ldap_result2error 42
  ldap_sasl_bind 32
  ldap_sasl_bind_s 32
  ldap_search 87
  ldap_search_ext_s 87
  ldap_search_s 87
  ldap_search_st 87
  ldap_server_conf_save 92
  ldap_server_free_list 92
  ldap_server_locate 92
  ldap_set_option 57
  ldap_set_option_np 57
  ldap_set_rebind_proc 32
  ldap_simple_bind 32
  ldap_simple_bind_s 32
  ldap_ssl_client_init 107
  ldap_ssl_init 107
  ldap_ssl_start 107
  ldap_unbind 57
  ldap_unbind_s 57
  ldap_url_parse 112
  ldap_url_search 112
  ldap_url_search_s 112
  ldap_url_search_st 112
  ldap_value_free 54
  ldap_value_free_len 54
  synchronous 4
routines, C utility 3
running operation utilities 115
running program 7

## S

sample Makefile 159
sample programs
  delete entries 161
  searching entries 169
  using LDAP API 161
sample socks.conf configuration file 6
SASL authentication 34
SASL GSS API bind 1
saving LDAP server information 92
schema
  modifying 129
search
  caching results 70
  control 25

search *(continued)*
  using LDAP 5
search response 85
search results, caching 17
search results, counting 50
searching entries 87
searching entries with timeout 87
searching for root DSE 140
searching for URL 112
searching for URL with timeout 112
Secure Sockets Layer (SSL)
  See SSL 107
security
  supported by LDAP 1
server authentication 108
server controls 24, 66
server, LDAP
  binding 32
  unbinding 32
service provider interface (SPI), LDAP 17
session controls 24
session settings 60
setting
  option 57
shell, z/OS 115
  running programs in 3
shortcut keys 183
SIGPIPE signal 109
SIGPIPE signals 9
simple authentication 1, 34
SOCKS server 5
SOCKS_CONF
  environmental variable 5
SOCKS_SERVER
  environmental variable 5
socksified client
  using 5
SPI (service provider interface), LDAP 17
SRV records 101
SSL (Secure Sockets Layer)
  ciphers 64
  creating connection 107
  inactivity timer 65
  initializing 107
standard error stream 14, 42
starting SSL 107
stepping through attributes 48
storage
  deallocating 73
  freeing 6
structure
  LDAP program 9
subentries, listing 13
Sun JNDI service provider 18
supported ciphers 64
synchronous LDAP operation 17
System SSL 110

## T

TCP/IP (Transmission Control Protocol/Internet Protocol)
  LDAP use of 19
termination functions 4
terms, glossary of 189
text message 67
textual data 67
thread safety
  LDAP API 17
Time Sharing Option (TSO)
  *See* TSO
TLS (Transport Layer Security) 1
trace classes 15
tracing
  disabling 6
  enabling 6, 15
Transmission Control Protocol/Internet Protocol (TCP/IP)
  *See* TCP/IP
Transport Layer Security (TLS) 1
TSO (Time Sharing Option)
  running operation utilities from 115
  using to run, link, compile 8
TXT records 101
typed
  name 2

## U

unbinding from LDAP server 32
unbinding LDAP API 10
Universal Resource Locator (URL)
  *See* URL
URL
  LDAP 58, 105, 107
URL (Universal Resource Locator)
  breaking down LDAP 112
  deallocating LDAP 112
  processing 5
  searching for LDAP 112
utilities, LDAP operation 115
utility functions
  LDAP 5
utility routines
  C 5

## W

waiting for result 85

## X

X.500, naming concepts 2
XDS/XOM 3

## Z

z/OS data sets 2
z/OS shell
  running operation utilities from 115

z/OS shell *(continued)*
  running programs 3

# Readers' Comments — We'd Like to Hear from You

**z/OS**
**Security Server**
**LDAP Client Programming**

**Publication No. SC24-5924-02**

**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**
SC24-5924-02

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

IBM Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY
12601-5400

Fold and Tape    **Please do not staple**    Fold and Tape

SC24-5924-02

Cut or Fold
Along Line

Program Number: 5694-A01, 5655-G52

Printed in U.S.A.

SC24-5924-02