

OS/390®



Security Server LDAP Client Application Development Guide and Reference

OS/390®



Security Server LDAP Client Application Development Guide and Reference

Note!

Before using this information and the product it supports, be sure to read the general information under Appendix A, "Notices" on page 109.

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.

IBM obtained the JNDI HTML documentation files from Sun Microsystems, Inc.

Second Edition (September 1999)

This edition, SC24-5878-01, applies to Version 2 Release 8 of OS/390 Security Server (5647-A01), and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SC24-5878-00.

© Copyright International Business Machines Corporation 1999. All rights reserved.

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

About This Book	vii
Who Should Use This Book	vii
How This Book Is Organized	vii
Conventions Used in This Book	vii
Where to Find More Information	viii
Online Books	viii
How to Send Your Comments	viii
Summary of Changes	ix
New Information for Release 8	ix
Changed Information for Release 8	ix
Chapter 1. LDAP Programming	1
How LDAP Is Defined	1
Data Model	2
LDAP Names	2
Function Overview	3
Compiling, Linking, and Running a Program	5
Using the API	6
Basic Structure	6
Performing an Operation	7
Example: Adding an entry	8
Example: Modifying an entry	8
Example: Deleting an entire entry	9
Example: Changing the RDN of an entry	9
Example: Comparing an attribute value with its value in an entry in the directory	9
Example: Reading a directory entry's contents	10
Example: Listing all sub-entries of an entry with associated objectClass attribute values	10
Example: Reading all objectClass attribute values for all entries below a given entry	10
Getting Results	10
Error Processing	11
Using ldap_get_errno() and ldap_result2error()	11
Using ldap_err2string() and ldap_perror()	12
Tracing	13
Threadafety	13
Synchronous Versus Asynchronous Operation	13
Example Program	14
LDAP Client for Java	20
Chapter 2. LDAP Routines	21
LDAP Controls	25
Session Controls	25
Using RACF® Data	26
Deprecated LDAP APIs	26
ldap_abandon	27
ldap_add	29
ldap_bind	32
ldap_compare	37
ldap_delete	40
ldap_error	42

ldap_first_attribute	46
ldap_first_entry/reference	48
ldap_get_dn	52
ldap_get_values	54
ldap_init	57
ldap_memfree	69
ldap_message	70
ldap_modify	72
ldap_parse_result	75
ldap_rename	77
ldap_result	80
ldap_search	83
ldap_ssl	88
ldap_url	92
LDAP Header Files	95
lber.h	95
ldap.h	96
ldapssl.h	105
Appendix A. Notices	109
Programming Interface Information	110
Trademarks	110
Glossary	111
Bibliography	115
IBM C/C++ Language Publication	115
IBM OS/390 Security Server Publication	115
IBM OS/390 Cryptographic Services Publication	115
Index	117

Tables

1.	LDAP API Functions	4
2.	LDAP Error Codes and Descriptions	43
3.	The optionValue Parameter Specifications	58

About This Book

This book describes the Lightweight Directory Access Protocol (LDAP) client application development for OS/390.

Who Should Use This Book

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

How This Book Is Organized

This book is divided into two chapters:

- Chapter 1, “LDAP Programming” on page 1 which describes how to use the LDAP client application programming interface.
- Chapter 2, “LDAP Routines” on page 21 which describes each LDAP client routine.

Conventions Used in This Book

This book uses the following typographic conventions:

Bold	Bold words or characters represent API names, attributes, status codes, environment variables, parameter values, and system elements that you must enter into the system literally, such as commands, options, or path names.
<i>Italic</i>	<i>Italic</i> 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 can repeat the preceding item one or more times.
\	A backslash is used as a continuation character when entering commands from the shell that exceed one line (255 characters). If the command exceeds one line, use the backslash character \ as the last nonblank character on the line to be continued, and continue the command on the next line.

Where to Find More Information

Where necessary, this book references information in other books, using shortened versions of the book title. For complete titles and order numbers of the books for all products that are part of OS/390, see the *OS/390 Information Roadmap*, GC28-1727. For a list of titles and order numbers of the books that are useful for the LDAP Client, see “Bibliography” on page 115.

Online Books

All the books belonging to the OS/390 DCE library are available as online publications. They are included in the *IBM OS/390 Collection*, SK2T-6700.

All the books in the Online Library are viewable, without charge, on these IBM operating platforms: OS/390, VM, OS/2®, DOS, and AIX/6000®. The same book can be viewed on any of these platforms using the IBM BookManager® Library Readers™ for OS/2, Windows, and DOS, or any of the IBM BookManager READ licensed programs for OS/390, VM, OS/2, Windows, DOS, or AIX/6000.

The booklet included with the Online Library provides details on accessing the OS/390 DCE online publications.

How to Send Your Comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this book or any other OS/390 documentation:

- Visit the home page at: <http://www.ibm.com/s390/os390>
- Fill out one of the forms at the back of the book and return it by mail, by fax, or by giving it to an IBM representative.

Summary of Changes

Summary of Changes

for SC24-5878-01

OS/390 Version 2 Release 8

This book contains information previously presented in *OS/390 Security Server LDAP Client Application Development Guide and Reference*, SC24-5878-00, which supports OS/390 Version 2 Release 7.

The following summarizes the changes to that information.

New Information for Release 8

- Two methods of security are supported: simple authentication and certificate authentication.
- The **LDAP_OPT_V2_WIRE_FORMAT** session option has been added.

Changed Information for Release 8

- The LDAP client and server are both packaged in the OS/390 Security Server and are always enabled.
- The information in “LDAP Client for Java” on page 20 has been removed and a reference to the online HTML version of the information is provided.
- The **ldapdelete.c** example program and the **lber.h**, **ldap.h**, and **ldapssl.h** header files have been updated.

This book includes terminology, maintenance, and editorial changes that are not marked. Technical changes or additions to the text and illustrations, however, are indicated by a vertical line to the left of the change.

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 OS/390 is for client access to Directory Services that accept the LDAP protocol. The LDAP client allows programs running on OS/390 UNIX to enter and extract information into and from a Directory Service. The OS/390 Security Server LDAP Server can be used to store and extract information on OS/390 using the LDAP protocol. See the *OS/390 Security Server LDAP Server Administration and Usage Guide*, SC24-5861 for more information.

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

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

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

This chapter focuses on the following topics:

- Defining the LDAP protocol
- The LDAP Data model, including the format of distinguished names in LDAP
- An overview of the functions supported by the LDAP client API on OS/390
- Details on compiling and link-editing a program that uses the LDAP client API
- Information on how to use the LDAP client APIs
- An example program which shows how the LDAP client API could be used as a Directory Service
- The LDAP Version 3 Client for Java

How LDAP Is Defined

The LDAP protocol is defined by a number of Internet Engineering Task Force (IETF) request for comments (RFCs). This protocol is defined in IETF RFC 1777 *Lightweight Directory Access Protocol*. Other RFCs of interest include:

- RFC 1778 - *The String Representation of Standard Attribute Syntaxes*
- RFC 1779 - *A String Representation of Distinguished Names*
- RFC 1960 - *A String Representation of LDAP Search Filters*
- RFC 1823 - *The LDAP Application Program Interface*
- RFC 2255 - *The LDAP URL Format*
- RFC 1738 - *Uniform Resource Locators (URL)*

IETF RFC 1823 defines a programming interface for using the LDAP protocol to communicate with a Directory Service that accepts the LDAP protocol. The programming interface available on OS/390 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 OS/390.

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

Data Model

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

To summarize, the directory is made up of entries. Each entry contains a set of attributes. These attributes can be single or multi-valued (have one or more values associated with them). The object class of an entry determines the set of attributes that must and the set of attributes that may exist in the entry. Refer to the *OS/390 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. Further, attribute types are provided to the program as null-terminated character strings instead of object identifiers.

LDAP Names

The LDAP protocol (and API) uses so-called “typed” names to identify directory entries. In contrast, DCE CDS and the Domain Name Service (DNS) use “untyped” names to identify entries. Each directory entry is identifiable by its fully distinguished name. The distinguished name (DN) is constructed by 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+1=Virginia, o=Acme International, c=US

An LDAP DN is specified as a null-terminated character string in a right-to-left fashion (right-to-left refers to the ordering of RDNs from highest to lowest in the directory hierarchy). Note that embedded spaces are taken as part of the attribute value for RDNs and do not require quotation marks. Also note that RDNs are separated by commas (,) and attribute type/value pairs within an RDN are separated by plus (+) signs. (Refer to IETF RFC 1779 for more information.)

Function Overview

The LDAP client API is provided in a C/C++ DLL that is loaded at run time by applications that use the LDAP API. Two forms of this DLL are provided, one for programs running under the OS/390 shell and one for programs running from OS/390 Datasets. Refer to “Compiling, Linking, and Running a Program” on page 5 for details on how to link-edit a program to use the proper form of the LDAP DLL.

Starting with OS/390 Release 7, the LDAP client DLLs are shipped in new locations in PDS and HFS.

- | The PDS versions of the DLLs are installed into **LPALIB**. The HFS versions are installed to **/usr/lpp/ldapclient/lib**. Symbolic links are set at installation to this new file system from **/usr/lib**, so HFS usage of the LDAP client files should be unchanged.

The LDAP API consists of 53 C language functions. All function names begin with the prefix **ldap_**. The functions can be broken down into six categories as shown in Table 1 on page 4.

For detailed information about each LDAP API see Chapter 2, “LDAP Routines” on page 21.

<i>Table 1. LDAP API Functions</i>	
Category	Function name
Initialization / Termination	ldap_init ldap_open ldap_ssl_init, ldap_ssl_client_init, ldap_ssl_start ldap_unbind, ldap_unbind_s
Primitive Operations	ldap_abandon ldap_add, ldap_add_s ldap_add_ext, ldap_add_ext_s ldap_bind, ldap_bind_s ldap_compare, ldap_compare_s ldap_compare_ext, ldap_compare_ext_s ldap_delete, ldap_delete_s ldap_delete_ext, ldap_delete_ext_s ldap_modify, ldap_modify_s ldap_modify_ext, ldap_modify_ext_s ldap_modrdn, ldap_modrdn_s ldap_rename, ldap_rename_s ldap_result ldap_sasl_bind, ldap_sasl_bind_s ldap_search, ldap_search_s, ldap_search_st ldap_search_ext, ldap_search_ext_s ldap_simple_bind, ldap_simple_bind_s
Error Handling	ldap_err2string ldap_get_errno ldap_perror ldap_result2error
Results Processing	ldap_count_attributes, ldap_first_attribute, ldap_next_attribute ldap_count_entries, ldap_first_entry, ldap_next_entry ldap_count_messages, ldap_count_references ldap_count_values, ldap_get_values ldap_count_values_len, ldap_get_values_len ldap_first_message, ldap_first_reference ldap_get_dn, ldap_get_entry_controls_np ldap_explode_dn ldap_msgid, ldap_msgtype ldap_next_message, ldap_next_reference ldap_parse_result, ldap_parse_reference_np ldap_parse_sasl_bind_result
LDAP URL Processing	ldap_is_ldap_url ldap_url_parse ldap_url_search, ldap_url_search_s, ldap_url_search_st
Utility Functions	ldap_control_free, ldap_controls_free ldap_memfree, ldap_msgfree ldap_mods_free, ldap_free_urldesc ldap_set_option, ldap_set_option_np, ldap_get_option ldap_set_rebind_proc ldap_value_free, ldap_value_free_len

Following is a description of each type of function:

Initialization and Termination Functions

The initialization functions are used to initialize the LDAP programming interface.

Primitive Operations

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

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

Error Handling Functions

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

Results Processing Functions

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

LDAP URL Processing Functions

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

Utility Functions

Utility functions are provided for freeing storage that was allocated by the LDAP API on behalf of the caller as well as for setting options that determine certain runtime characteristics of the LDAP programming interface. An example of an option that can be set is the debug level which allows tracing to be selectively enabled and disabled at run time.

Compiling, Linking, and Running a Program

As previously stated, the LDAP programming interface is provided as a C/C++ DLL. The DLL 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. 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 OS/390 shell or to run from a PDS, this exports file should be specified as **/usr/lib/GLDCLDAP.x**.

Note: Release 7 of the LDAP Server was the last release in which **EUVCLDAP.x**, **EUVCLDAP**, **ldap.x**, and **ldap.dll** were available. Applications should use **GLDCLDAP.x** and **GLDCLDAP** during link-edit and at run time.

5. When running the program, be sure that the LDAP DLL is accessible. When running under the OS/390 shell, be sure that the LIBPATH environment variable includes **/usr/lib**. When running the program from an OS/390 dataset, the DLLs will be found in **LPALIB**.

6. If using SSL, follow these steps:

a. Put

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

in all C or C++ source files.

b. Ensure that **STEPLIB** or **LIBPATH** identifies the SGSKLOAD DLL.

c. Specify **/usr/lib/GSKSSL.x** in the link-edit.

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(LAPDLET)'" ldapdelete.o /usr/lib/GLDCLDAP.x
             touch LAPDLET
```

Using the API

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

Basic Structure

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

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

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

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

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

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

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

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

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

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

Examples of calls to perform LDAP operations are provided in “Performing an Operation.” See “Getting Results” on page 10 for examples of calls to **ldap_result()** as well as calls to the LDAP results processing routines. When using LDAP Version 3 protocol, **ldap_parse_result** can be used.

4. When all LDAP operations are completed, unbind and de-initialize the LDAP programming interface using **ldap_unbind()** or **ldap_unbind_s()**. Note that **ldap_unbind_s()** is identical in function to **ldap_unbind()**. It is provided as a convenience to those programs that only do synchronous operations so that the unbind does not appear to be an asynchronous operation. All unbind operations are synchronous. Also note that after the **ldap_unbind()** or **ldap_unbind_s()** function returns, the LDAP handle that was returned by **ldap_open()** or **ldap_init()** is no longer valid and must not be used.

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

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

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

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

Performing an Operation

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

An example call to each LDAP operation will be shown here along with a short explanation of each parameter's meaning. Refer to Chapter 2, “LDAP Routines” on page 21 for details on the parameters to each LDAP function in the LDAP API.

Example: Adding an entry

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

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

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

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

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

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

Example: Modifying an entry

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

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

modifications[1]->mod_op = LDAP_MOD_ADD;
modifications[1]->mod_type = "email";
modifications[1]->mod_values = (char **)malloc( sizeof(char *)*2 );
```

```

modifications[1]->mod_values[0] = "johnd@acme.com";
modifications[1]->mod_values[1] = NULL;

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

```

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

Example: Deleting an entire entry

```

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

```

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

Example: Changing the RDN of an entry

```

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

```

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

Example: Comparing an attribute value with its value in an entry in the directory

```

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

```

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

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

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

Example: Reading a directory entry's contents

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

Example: Listing all sub-entries of an entry with associated objectClass attribute values

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

Example: Reading all objectClass attribute values for all entries below a given entry

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

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

Getting Results

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

An example of this type of processing is:

```

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

```

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

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

Error Processing

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

Using **ldap_get_errno()** and **ldap_result2error()**

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

There is a subtle difference between using **ldap_get_errno()** and **ldap_result2error()** for asynchronous operations. For asynchronous operations, if an error occurs during the process of sending the request to the directory server, you must use **ldap_get_errno()** to obtain the error value. Use the **ldap_result2error()** call after a **ldap_result()** call has completed. In the case of synchronous operations,

either function can be used. In addition, the synchronous functions also return the error code value for the programmer's convenience.

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

Example: Retrieving the error code of an asynchronous operation request

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

    /* process the error */

}
```

Example: Retrieving the error code using ldap_result2error()

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

    /* process the error */

}
```

Using ldap_err2string() and ldap_perror()

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

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

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

Example: Obtaining and using 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 in a message or log file entry */

}
```

Example: Sending the result of an operation to the standard error stream

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

Tracing

Tracing can be enabled in the LDAP programming interface. This is done by one of two methods. The first method is to use the **ldap_set_option()** API, specifying the option to be set as **LDAP_OPT_DEBUG**. The second method for enabling tracing is to set the **LDAP_DEBUG** environment variable. The value for **LDAP_DEBUG** should be an integer based on the set of trace classes that the user wishes to enable. Consult the “Debug Levels” section of the **ldap.h** header file (“ldap.h” on page 96) for a specification of these trace classes. Note that the **LDAP_DEBUG** environment variable can be used without recompiling the client program and provides a means of enabling tracing without changing the client program. The **ldap_set_option()** call can be used for limiting the areas of client program operation that should be traced. Trace output is put on the standard error stream.

An example of enabling all trace classes using the **LDAP_DEBUG** environment variable (assuming the program is running from the OS/390 shell) is to enter:

```
export LDAP_DEBUG=65535
```

on the OS/390 shell command line prior to running the client program. An example of enabling all trace classes using the **ldap_set_option()** LDAP API is:

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

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

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

Threadsaafety

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

Synchronous Versus Asynchronous Operation

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

However, there is some interplay with the threadsafety 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 multithreaded environment.

With synchronous operations, even though multiple operations can be initiated on separate threads, the threadsafety 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.

Example Program

The following example program 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.

```
| ??=ifdef __COMPILER_VER__
| ??=pragma filetag ("IBM-1047")
| ??=endif

| /*****
| /* THIS FILE CONTAINS SAMPLE CODE.  IBM PROVIDES THIS CODE ON AN      */
| /* 'AS IS' BASIS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS        */
| /* OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES   */
| /* OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.           */
| /*****/

| /*
|  * Copyright (c) 1995 Regents of the University of Michigan.
|  * All rights reserved.
|  *
|  * Redistribution and use in source and binary forms are permitted
|  * provided that this notice is preserved and that due credit is given
|  * to the University of Michigan at Ann Arbor. The name of the University
|  * may not be used to endorse or promote products derived from this
|  * software without specific prior written permission. This software
|  * is provided "as is" without express or implied warranty.
|  */

| /* ldapdelete.c - simple program to delete an entry using LDAP */

| #include <stdio.h>
| #include <string.h>
| #include <strings.h>
| #include <stdlib.h>
| #include <ctype.h>
| #include <ldap.h>
| #include <locale.h>

| #ifndef TRUE
|     #define TRUE 1
| #endif

| #ifndef FALSE
```

```

|     #define FALSE 0
| #endif

| static LDAP *ld;
| static char *prog;
| static char *binddn = NULL;
| static char *passwd = NULL;
| static char *ldaphost = "localhost";
| static int ldapport = LDAP_PORT;
| static int not = FALSE;
| static int verbose = FALSE;
| static int contoper = FALSE;
| static int follow_referrals = LDAP_OPT_ON;
| static int deref = LDAP_DEREF_NEVER;
| static int ldapversion = LDAP_VERSION2;
| static int manageDsa = FALSE;
| static LDAPControl manageDsaIT = {
|     "2.16.840.1.113730.3.4.2", /*OID*/
|     { 0, NULL }, /*no value*/
|     LDAP_OPT_ON /*critical*/
| };
| static LDAPControl *M_controls[2] = { &manageDsaIT, NULL};

| static void usage( char *s );
| static int dodelete( LDAP *ld, char *dn );
| int rebindproc( LDAP *ld, char **dn, char **pwp, int *methodp, int freeit );

| main( int argc, char **argv )
| {
|     char *optpattern = "nvRMZc?h:V:p:D:w:d:f:K:P:N:";
|     int ssl = FALSE;
|     char *keyfile = NULL, *keyfile_pw = NULL, *keyfile_dn = NULL;
|     char *p, buf[ 4096 ];
|     FILE *fp;
|     int i, rc=LDAP_SUCCESS, port = FALSE;
|     int debugLevel = 0;
|     int debugSpecified = FALSE;
|     int failureReasonCode ;
|     extern char *optarg;
|     extern int optind;

|     setlocale( LC_ALL, "" );

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

|     not = verbose = contoper = ssl = port = FALSE;
|     fp = NULL;

|     while ( ( i = getopt( argc, argv, optpattern ) ) != EOF ) {
|         switch ( i ) {

```

```

| case 'V':
|     ldapversion = atoi( optarg );
|     if ( ldapversion != LDAP_VERSION2 &&
|         ldapversion != LDAP_VERSION3 ) {
|         fprintf( stderr, "Incorrect version level supplied.\n");
|         fprintf( stderr, "Supported values for the -V parameter"
|             " are 2 and 3\n");
|         exit( 1 );
|     }
|     break ;
| case 'c': /* continue even if error encountered */
|     contoper = TRUE;
|     break;
| case 'h': /* ldap host */
|     ldaphost = strdup( optarg );
|     break;
| case 'D': /* bind DN */
|     binddn = strdup( optarg );
|     break;
| case 'w': /* password */
|     passwd = strdup( optarg );
|     break;
| case 'f': /* read DN's from a file */
|     if ( ( fp = fopen( optarg, "r" ) ) == NULL ) {
|         perror( optarg );
|         exit( 1 );
|     }
|     break;
| case 'd':
|     debugLevel = atoi( optarg );
|     debugSpecified = TRUE;
|     break;
| case 'p':
|     ldapport = atoi( optarg );
|     port = TRUE;
|     break;
| case 'n': /* print deletes, don't actually do them */
|     not = TRUE;
|     break;
| case 'R': /* don't automatically chase referrals */
|     follow_referrals = LDAP_OPT_OFF;
|     break;
| case 'M':
|     manageDsa = TRUE;
|     break;
| case 'v': /* verbose mode */
|     verbose = TRUE;
|     break;
| case 'K':
|     keyfile = strdup( optarg );
|     break;
| case 'P':
|     keyfile_pw = strdup( optarg );
|     break;
| case 'N':
|     keyfile_dn = strdup( optarg );
|     break;
| case 'Z':

```

```

|         ssl = TRUE;
|         break;
|     case '?':
|     default:
|         usage( prog );
|         exit( 1 );
|     }
| }

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

| if ( fp == NULL ) {
|     if ( optind >= argc ) {
|         fp = stdin;
|     }
| }

| if ( !not ) {
|     if ( ssl ) {
|         if ( !port ) {
|             ldapport = LDAPS_PORT;
|         }

|         if ( keyfile == NULL ) {
|             keyfile = getenv( "SSL_KEYRING" );
|             if ( keyfile != NULL ) {
|                 keyfile = strdup(keyfile);
|             }
|         }

|         if (verbose) {
|             printf( "ldap_ssl_client_init( %s, %s, 0,"
|                 " &failureReasonCode )\n",
|                 keyfile ? keyfile : "NULL",
|                 keyfile_pw ? keyfile_pw : "NULL" );
|         }
|         rc = ldap_ssl_client_init( keyfile, keyfile_pw, 0,
|                                   &failureReasonCode );
|         if ( rc != LDAP_SUCCESS ) {
|             fprintf( stderr,
|                 "ldap_ssl_client_init failed! rc == %d,"
|                 " failureReasonCode == %d\n",
|                 rc, failureReasonCode );
|             exit( 1 );
|         }
|         if ( verbose ) {
|             printf( "ldap_ssl_init( %s, %d, %s )\n",
|                 ldaphost, ldapport,
|                 keyfile_dn ? keyfile_dn : "NULL" );
|         }
|         ld = ldap_ssl_init( ldaphost, ldapport, keyfile_dn );
|         if ( ld == NULL ) {
|             fprintf( stderr, "ldap_ssl_init failed\n" );
|             perror( ldaphost );
|             exit( 1 );
|         }
|     }
| }

```

```

|         }
|     }
|     else {
|         if ( verbose ) {
|             printf( "ldap_init(%s, %d) \n", ldaphost, ldapport );
|         }
|         if ( ( ld = ldap_init( ldaphost, ldapport ) ) == NULL ) {
|             perror( ldaphost );
|             exit( 1 );
|         }
|     }

|     ldap_set_option_np( ld, LDAP_OPT_PROTOCOL_VERSION, ldapversion );
|     if ( debugSpecified ) {
|         ldap_set_option_np( ld, LDAP_OPT_DEBUG, debugLevel );
|     }
|     ldap_set_option_np( ld, LDAP_OPT_DEREF, deref );
|     ldap_set_option_np( ld, LDAP_OPT_REFERRALS, follow_referrals );

|     if ( binddn != NULL ) {
|         ldap_set_rebind_proc( ld, (LDAPRebindProc)rebindproc );
|     }

|     if ( ldapversion == LDAP_VERSION2 && binddn != NULL ) {
|         /*
|          * Bind is required for LDAP V2 protocol,
|          * but not for V3 (or later) protocols.
|          * We also bind if a bind DN was specified.
|          */
|         if ( ldap_bind_s( ld, binddn, passwd, LDAP_AUTH_SIMPLE )
|             != LDAP_SUCCESS ) {
|             ldap_perror( ld, "ldap_bind" );
|             exit( 1 );
|         }
|     }
| } /* ! not */

| if ( fp == NULL ) {
|     for ( ; (rc == LDAP_SUCCESS] contoper) && optind < argc; ++optind ) {
|         rc = dodelete( ld, argv[ optind ] );
|     }
| }
| else {
|     rc = LDAP_SUCCESS;
|     while ( (rc == LDAP_SUCCESS] contoper) &&
|         fgets( buf, sizeof(buf), fp ) != NULL ) {
|         buf[ strlen( buf ) - 1 ] = '\0';    /* remove trailing newline */
|         if ( *buf != '\0' ) {
|             rc = dodelete( ld, buf );
|         }
|     }
| }

| if ( !not ) {
|     ldap_unbind( ld );
| }

```

```

    exit( rc );
}

static void usage( char *s )
{
    fprintf( stderr, "usage: %s [options] [ -f file ] < entryfile ] dn ... ]\n",
             s );
    fprintf( stderr, "where:\n" );
    fprintf( stderr, "    dn\tdistinguished name of entry to delete\n" );
    fprintf( stderr, "    entryfile\tfile containing DN's to delete\n" );
    fprintf( stderr, "    \t\ton consecutive lines\n" );
    fprintf( stderr, "options:\n" );
    fprintf( stderr, "    -?\t\tprint this text\n" );
    fprintf( stderr, "    -V version\tselect LDAP protocol version"
             " (2 or 3; default is 2)\n" );
    fprintf( stderr, "    -c\t\tcontinue even if error encountered\n" );
    fprintf( stderr, "    -n\t\tshow what would be done but don't actually"
             " delete\n" );
    fprintf( stderr, "    -v\t\trun in verbose mode (diagnostics to"
             " standard output)\n" );
    fprintf( stderr, "    -R\t\tdo not automatically follow referrals\n" );
    fprintf( stderr, "    -M\t\tTreat referral objects as normal entries."
             " (requires -V 3)\n" );
    fprintf( stderr, "    -d level\tset LDAP debugging level to 'level'\n" );
    fprintf( stderr, "    -f file\tperform sequence of deletes listed"
             " in 'file'\n" );
    fprintf( stderr, "    -D binddn\tbind dn\n" );
    fprintf( stderr, "    -w passwd\tbind passwd (for simple"
             " authentication)\n" );
    fprintf( stderr, "    -h host\tldap server\n" );
    fprintf( stderr, "    -p port\tport on ldap server\n" );
    fprintf( stderr, "    -Z\t\tuse a secure ldap connection for the"
             " operation\n" );
    fprintf( stderr, "    -K keyfile\tfile to use for keys/certificates\n" );
    fprintf( stderr, "    -P key_pw\tkeyfile password\n" );
    fprintf( stderr, "    -N key_dn\tCertificate Name in keyfile\n" );
}

static int dodelete( LDAP *ld, char *dn )
{
    int rc;

    if ( verbose ) {
        printf( "%sdeleting entry %s\n", not ? "!" : "", dn );
    }
    if ( not ) {
        rc = LDAP_SUCCESS;
    }
    else {
        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 );
| }

```

LDAP Client for Java

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

Chapter 2. LDAP Routines

This chapter describes the Lightweight Directory Access Protocol (LDAP) routines which are grouped according to function. The LDAP routines provide access through TCP/IP to directory services which accept the LDAP protocol. Also, the following header files are shown in this chapter:

- “lber.h” on page 95
- “ldap.h” on page 96
- “ldaps.h” on page 105

For information on how to use the LDAP functions, including an example program, see Chapter 1, “LDAP Programming” on page 1. For information on the LDAP Server, see *OS/390 Security Server LDAP Server Administration and Usage Guide*.

Following is a summary of the LDAP routines:

ldap_abandon	Abandons an asynchronous LDAP operation that is in progress. (See “ldap_abandon” on page 27.)
ldap_abandon_ext	Abandons an asynchronous operation with controls. (See “ldap_abandon” on page 27.)
ldap_add	Performs an asynchronous LDAP add operation. (See “ldap_add” on page 29.)
ldap_add_ext	Performs an asynchronous LDAP add operation with controls. (See “ldap_add” on page 29.)
ldap_add_ext_s	Performs a synchronous LDAP add operation with controls. (See “ldap_add” on page 29.)
ldap_add_s	Performs a synchronous LDAP add operation. (See “ldap_add” on page 29.)
ldap_bind	Binds to an LDAP server asynchronously in order to perform directory operations. (See “ldap_bind” on page 32.)
ldap_bind_s	Binds to an LDAP server synchronously in order to perform directory operations. (See “ldap_bind” on page 32.)
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 69.)
ldap_controls_free	Frees an array of LDAPControl structures. (See “ldap_memfree” on page 69.)
ldap_count_attributes	Counts the number of attributes in an entry returned as part of a search result. (See “ldap_first_attribute” on page 46.)
ldap_count_entries	Retrieves a count of the entries in a chain of search results. (See “ldap_first_entry/reference” on page 48.)

ldap_count_messages	Returns the number of messages in a result chain, as returned by ldap_result . (See “ldap_message” on page 70.)
ldap_count_references	Returns the number of continuation references in a chain of search results. (See “ldap_first_entry/reference” on page 48.)
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 40.)
ldap_delete_ext	Performs an asynchronous LDAP delete operation with controls. (See “ldap_delete” on page 40.)
ldap_delete_ext_s	Performs a synchronous LDAP delete operation with controls. (See “ldap_delete” on page 40.)
ldap_delete_s	Performs a synchronous LDAP delete operation. (See “ldap_delete” on page 40.)
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 52.)
ldap_first_attribute	Begins stepping through an LDAP entry's attributes. (See “ldap_first_attribute” on page 46.)
ldap_first_entry	Retrieves the first entry in a chain of search results. (See “ldap_first_entry/reference” on page 48.)
ldap_first_message	Retrieves the first message in a result chain, as returned by ldap_result . (See “ldap_message” on page 70.)
ldap_first_reference	Retrieves the first continuation reference in a chain of search results. (See “ldap_first_entry/reference” on page 48.)
ldap_free_urldesc	Deallocates an LDAP URL description obtained from a call to ldap_url_parse . (See “ldap_url” on page 92.)
ldap_get_dn	Obtains LDAP distinguished names from an LDAP entry. (See “ldap_get_dn” on page 52.)
ldap_get_entry_controls_np	Extracts server controls from an entry. (See “ldap_first_entry/reference” on page 48.)
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 92.)
ldap_memfree	Deallocates character strings allocated by the LDAP programming interface. (See “ldap_memfree” on page 69.)
ldap_modify	Performs an asynchronous LDAP modify operation. (See “ldap_modify” on page 72.)
ldap_modify_ext	Performs an asynchronous LDAP modify operation with controls. (See “ldap_modify” on page 72.)
ldap_modify_ext_s	Performs a synchronous LDAP modify operation with controls. (See “ldap_modify” on page 72.)
ldap_modify_s	Modifies LDAP entries synchronously. (See “ldap_modify” on page 72.)
ldap_modrdn	Performs an asynchronous LDAP modify relative distinguished name (RDN) operation. (See “ldap_rename” on page 77.)
ldap_modrdn_s	Performs a synchronous LDAP modify RDN operation. (See “ldap_rename” on page 77.)
ldap_mods_free	Deallocates a NULL-terminated array of modification structures. (See “ldap_modify” on page 72.)
ldap_msgfree	Deallocates the memory allocated for a result. (See “ldap_result” on page 80.)
ldap_msgid	Retrieves the message ID associated with an LDAP message. (See “ldap_result” on page 80.)
ldap_msgtype	Retrieves the type of an LDAP message. (See “ldap_result” on page 80.)
ldap_next_attribute	Retrieves the next attribute type name in an LDAP result. (See “ldap_first_attribute” on page 46.)
ldap_next_entry	Retrieves the next entry in a chain of search results to parse. (See “ldap_first_entry/reference” on page 48.)
ldap_next_message	Retrieves the next message in a result chain, as returned by ldap_result . (See “ldap_message” on page 70.)
ldap_next_reference	Retrieves the next continuation reference in a chain of search results. (See “ldap_first_entry/reference” on page 48.)
ldap_open	Initializes an LDAP context and opens a connection to an LDAP server under that context. (See “ldap_init” on page 57.)
ldap_parse_reference_np	Extracts information from a continuation reference. (See “ldap_first_entry/reference” on page 48.)
ldap_parse_result	Extracts information from results. (See “ldap_parse_result” on page 75.)
ldap_parse_sasl_bind_result	Extracts server credentials from SASL bind results. (See “ldap_parse_result” on page 75.)
ldap_perror	Prints an indication of the error on the standard error stream. (See “ldap_error” on page 42.)
ldap_rename	Performs an asynchronous LDAP rename operation. (See “ldap_rename” on page 77.)

ldap_rename_s	Performs a synchronous LDAP rename operation. (See “ldap_rename” on page 77.)
ldap_result	Waits for the result of an LDAP operation. (See “ldap_result” on page 80.)
ldap_result2error	Interprets a result as returned by ldap_result or one of the synchronous LDAP search operation routines. (See “ldap_error” on page 42.)
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 83.)
ldap_search_ext	Performs an asynchronous LDAP search operation with controls. (See “ldap_search” on page 83.)
ldap_search_ext_s	Performs a synchronous LDAP search operation with controls. (See “ldap_search” on page 83.)
ldap_search_s	Performs a synchronous LDAP search operation. (See “ldap_search” on page 83.)
ldap_search_st	Performs a synchronous LDAP search operation allowing a timeout to be specified to limit the time to wait for results. (See “ldap_search” on page 83.)
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 88.)
ldap_ssl_init	Initializes an SSL connection. (See “ldap_ssl” on page 88.)
ldap_ssl_start	Creates a secure SSL connection. (See “ldap_ssl” on page 88.)
ldap_unbind	Unbinds from an LDAP server asynchronously and deallocates an LDAP handle. (See “ldap_bind” on page 32.)
ldap_unbind_s	Unbinds from an LDAP server synchronously and deallocates an LDAP handle. (See “ldap_bind” on page 32.)
ldap_url_parse	Breaks down an LDAP URL into its component pieces. (See “ldap_url” on page 92.)
ldap_url_search	Initiates an asynchronous LDAP search based on an LDAP URL. (See “ldap_url” on page 92.)
ldap_url_search_s	Initiates a synchronous LDAP search based on an LDAP URL. (See “ldap_url” on page 92.)

ldap_url_search_st	Initiates a synchronous LDAP search based on an LDAP URL allowing a timeout to be specified to limit the time to wait for results. (See “ldap_url” on page 92.)
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. Note that client-side controls are not defined for this client library. A common data structure is used to represent both server-side and client-side controls:

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

The LDAPControl fields have the following definitions:

<i>ldctl_oid</i>	Specifies the control type, represented as a string.
<i>ldctl_value</i>	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.
<i>ldctl_iscritical</i>	Specifies whether the control is critical. If this field is nonzero (critical), the operation is performed only if the control is appropriate for the operation and it is recognized and supported by the server (or the client for client-side controls). In this case, the control is used in performing the operation. If this field is zero (noncritical), the control is used in performing the operation only if it is appropriate for the operation and it is recognized and supported by the server (or the client for client-side controls). Otherwise, the control will be ignored.

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

Session Controls

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

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

Using RACF® Data

There are some restrictions when updating information stored in RACF over the LDAP protocol. See the information about accessing RACF information in the *OS/390 Security Server LDAP Server Administration and Usage Guide*.

Deprecated LDAP APIs

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

- **ldap_ssl_start** (use **ldap_ssl_client_init** and **ldap_ssl_init**)
- **ldap_open** (use **ldap_init**)
- **ldap_bind** (use **ldap_simple_bind**)
- **ldap_bind_s** (use **ldap_simple_bind_s**)
- **ldap_modrdn** (use **ldap_rename**)
- **ldap_modrdn_s** (use **ldap_rename_s**)
- **ldap_result2error** (use **ldap_parse_result**)
- **ldap_perror** (use **ldap_parse_result**)

ldap_abandon

ldap_abandon
ldap_abandon_ext

Purpose

Abandon an asynchronous LDAP operation that is in progress.

Format

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

```
int ldap_abandon(  
    LDAP *ld,  
    int msgid)
```

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

Parameters

Input

<i>ld</i>	Specifies the LDAP handle returned by a previous call to ldap_open , ldap_ssl_init , or ldap_init .
<i>msgid</i>	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.
<i>serverctrls</i>	Specifies a list of LDAP server controls. This parameter may be set to NULL. See “LDAP Controls” on page 25 for more information about server controls.
<i>clientctrls</i>	Specifies a list of LDAP client controls. This parameter may be set to NULL. See “LDAP 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

<i>ld</i>	Specifies the LDAP handle returned by a previous call to ldap_open , ldap_ssl_init , or ldap_init .
<i>dn</i>	Specifies the distinguished name of the entry to add.
<i>attrs</i>	A NULL-terminated array of the entry's attributes. The LDAPMod structure is used to represent attributes, with the <i>mod_type</i> and <i>mod_values</i> fields being used as described under ldap_modify , and the <i>mod_op</i> field being used only if you need to specify the LDAP_MOD_BVALUES option.

ldap_add

Otherwise, it should be set to 0. The LDAPMod structure is shown in “ldap_modify” on page 72.

serverctrls Specifies a list of LDAP server controls. This parameter may be set to NULL. See “LDAP Controls” on page 25 for more information about server controls.

clientctrls Specifies a list of LDAP client controls. This parameter may be set to NULL. See “LDAP Controls” on page 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 OS/390 LDAP Server running with an SDBM database, the **ldap_add** APIs can return **LDAP_OTHER** and have completed a partial update to an entry in RACF. The results will match what would occur if the update were done using the RACF **altuser** command. If several RACF attributes are being updated and one of them is in error, RACF reports on the error, but still updates the other attributes. The RACF message text is also returned in the result.

Related Topics

[ldap_modify](#)

ldap_bind

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

Purpose

LDAP routines for binding and unbinding.

Format

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

```
int ldap_sasl_bind(  
    LDAP *ld,  
    char *who,  
    char *mechanism,  
    struct berval *cred,  
    LDAPControl **serverctrls,  
    LDAPControl **clientctrls,  
    int *msgidp)  
  
int ldap_sasl_bind_s(  
    LDAP *ld,  
    char *who,  
    char *mechanism,  
    struct berval *cred,  
    LDAPControl **serverctrls,  
    LDAPControl **clientctrls,  
    struct berval **servercredp)  
  
int ldap_simple_bind(  
    LDAP *ld,  
    char *who,  
    char *passwd)  
  
int ldap_simple_bind_s(  
    LDAP *ld,  
    char *who,  
    char *passwd)  
  
int ldap_unbind(  
    LDAP *ld)  
  
int ldap_unbind_s(  
    LDAP *ld)  
  
void ldap_set_rebind_proc(  
    LDAP *ld,
```

LDAPRebindProc *rebindproc*)

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

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

Parameters

Input

<i>ld</i>	Specifies the LDAP handle returned by a previous call to ldap_open , ldap_ssl_init , or ldap_init .
<i>who</i>	Specifies the distinguished name of the entry as which to bind.
<i>cred</i>	Specifies the password used in association with the DN of the entry (<i>who</i>) 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 <i>mechanism</i> parameter.
<i>mechanism</i>	Although a variety of mechanisms have been IANA (Internet Assigned Numbers Authority) registered, the only mechanism supported by the library at this time is the LDAP_MECHANISM_EXTERNAL mechanism, represented by the string LDAP_MECHANISM_EXTERNAL . The LDAP_MECHANISM_EXTERNAL mechanism indicates to the server that information external to SASL should be used to determine whether the client is authorized to authenticate. For this implementation, the system providing the external information must be SSL. For example, if the client sets <i>dn</i> and <i>credential</i> to NULL (the value of the pointers should be NULL), with <i>mechanism</i> set to LDAP_MECHANISM_EXTERNAL , the client is requesting that the server use the strongly authenticated identity to access the directory.
<i>method</i>	Selects the authentication method to use. Specify LDAP_AUTH_SIMPLE for simple authentication. (Simple authentication is the only supported method.)
<i>passwd</i>	Specifies the password used in association with the DN of the entry as which to bind.
<i>serverctrls</i>	Specifies a list of LDAP server controls. This parameter may be set to NULL. See “LDAP Controls” on page 25 for more information about server controls.
<i>clientctrls</i>	Specifies a list of LDAP client controls. This parameter may be set to NULL. See “LDAP Controls” on page 25 for more information about client controls.

ldap_bind

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

Output

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

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

Usage

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

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

Simple Authentication

The simplest form of the bind call is the synchronous API **ldap_simple_bind_s**. It takes the DN to bind as, as well as the password associated with that DN (supplied in *passwd*). It returns an LDAP error indication (see “ldap_error” on page 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 and certificate authentication over LDAP through the use of the Simple Authentication Security Layer (SASL). By setting *mechanism* to **LDAP_SASL_SIMPLE** the SASL bind request will be interpreted as a request for simple authentication (that is, equivalent to using **ldap_simple_bind** or **ldap_simple_bind_s**). By setting *mechanism* to **LDAP_MECHANISM_EXTERNAL**, the SASL bind request will be interpreted as a request for certificate authentication.

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

1. **ldap_ssl_client_init** (initialize the SSL library)
2. **ldap_ssl_init** (*host, port, name*), where *name* references a public/private key pair in the client's key ring file

3. `ldap_sasl_bind_s` (*ld*, *who*=NULL, *mechanism*=LDAP_MECHANISM_EXTERNAL, *cred*=NULL...)

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

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

Unbinding

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

Rebinding While Following Referrals

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

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

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

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

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

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

Error Conditions

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

ldap_error

ldap_compare

ldap_compare
 ldap_compare_s
 ldap_compare_ext
 ldap_compare_ext_s

Purpose

Perform an LDAP compare operation.

Format

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

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

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

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

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

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

Parameters

Input

<i>ld</i>	Specifies the LDAP handle returned by a previous call to ldap_open , ldap_ssl_init , or ldap_init .
<i>dn</i>	Specifies the distinguished name of the entry upon which to perform the compare.
<i>attr</i>	Specifies the attribute type to compare to the attribute found in the entry.
<i>bvalue</i>	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.
<i>value</i>	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.
<i>serverctrls</i>	Specifies a list of LDAP server controls. This parameter may be set to NULL. See “LDAP Controls” on page 25 for more information about server controls.
<i>clientctrls</i>	Specifies a list of LDAP client controls. This parameter may be set to NULL. See “LDAP Controls” on page 25 for more information about client controls.

Output

<i>msgidp</i>	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

<i>ld</i>	Specifies the LDAP handle returned by a previous call to ldap_open , ldap_ssl_init , or ldap_init .
<i>dn</i>	Specifies the distinguished name of the entry to be deleted.
<i>serverctrls</i>	Specifies a list of LDAP server controls. This parameter may be set to NULL. See “LDAP Controls” on page 25 for more information about server controls.
<i>clientctrls</i>	Specifies a list of LDAP client controls. This parameter may be set to NULL. See “LDAP 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 delete 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_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

<i>ld</i>	Specifies the LDAP handle returned by a previous call to ldap_open , ldap_ssl_init , or ldap_init .
<i>s</i>	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 .
<i>res</i>	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 83).
<i>freeit</i>	Specifies whether to deallocate the <i>res</i> LDAP result. If nonzero, the <i>res</i> parameter is deallocated as part of the call to ldap_result2error .
<i>err</i>	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 multithreaded 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 multithreaded applications, the error text printed corresponds to the last error value set by the last LDAP operation performed against the LDAP handle. It is possible for an LDAP operation on a different thread to reset the error value stored under the LDAP handle before the original error text is retrieved.

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

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

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

Error Conditions

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

Table 2 (Page 1 of 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.

Table 2 (Page 2 of 3). LDAP Error Codes and Descriptions

Value	Text (English version)	Detailed description
LDAP_UNAVAILABLE_CRITICAL_EXTENSION	Critical extension not supported	Critical extension is not supported.
LDAP_CONFIDENTIALITY_REQUIRED	Confidentiality is required	Confidentiality is required.
LDAP_SASLBIND_IN_PROGRESS	SASL bind in progress	An SASL bind is in progress.
LDAP_NO_SUCH_ATTRIBUTE	No such attribute	The attribute type specified does not exist in the entry.
LDAP_UNDEFINED_TYPE	Undefined attribute type	The attribute type specified is not valid.
LDAP_INAPPROPRIATE_MATCHING	Inappropriate matching	Filter type not supported for the specified attribute.
LDAP_CONSTRAINT_VIOLATION	Constraint violation	An attribute value specified violates some constraint (for example, a postalAddress 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 userPassword attribute).
LDAP_INVALID_CREDENTIALS	Invalid credentials	Invalid credentials were presented (for example, the wrong password).
LDAP_INSUFFICIENT_ACCESS	Insufficient access	The user has insufficient access to perform the operation.
LDAP_BUSY	DSA is busy	The DSA is busy.
LDAP_UNAVAILABLE	DSA is unavailable	The DSA is unavailable.
LDAP_UNWILLING_TO_PERFORM	DSA is unwilling to perform	The DSA is unwilling to perform the operation.
LDAP_LOOP_DETECT	Loop detected	A loop was detected.
LDAP_NAMING_VIOLATION	Naming violation	A naming violation occurred.
LDAP_OBJECT_CLASS_VIOLATION	Object class violation	An object class violation occurred (for example, a "required" attribute was missing from the entry).
LDAP_NOT_ALLOWED_ON_NONLEAF	Operation not allowed on nonleaf	The operation is not allowed on a nonleaf object.
LDAP_NOT_ALLOWED_ON_RDN	Operation not allowed on RDN	The operation is not allowed on an RDN.
LDAP_ALREADY_EXISTS	Already exists	The entry already exists.
LDAP_NO_OBJECT_CLASS_MODS	Cannot modify object class	Object class modifications are not allowed.
LDAP_RESULTS_TOO_LARGE	Results too large	Results too large.
LDAP_AFFECTS_MULTIPLE_DSAS	Affects multiple DSAs	Affects multiple DSAs.
LDAP_OTHER	Unknown error	An unknown error occurred.
LDAP_SERVER_DOWN	Can't contact LDAP server	The LDAP library cannot contact the LDAP server.
LDAP_LOCAL_ERROR	Local error	Some local error occurred. This is usually a failed memory allocation.
LDAP_ENCODING_ERROR	Encoding error	An error was encountered encoding parameters to send to the LDAP server.
LDAP_DECODING_ERROR	Decoding error	An error was encountered decoding a result from the LDAP server.

Table 2 (Page 3 of 3). LDAP Error Codes and Descriptions

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

Related Topics

[ldap_memfree](#)

[ldap_parse_result](#)

ldap_first_attribute

ldap_count_attributes
 ldap_first_attribute
 ldap_next_attribute

Purpose

Step through LDAP entry attributes.

Format

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

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

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

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

Parameters

Input

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

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

Output

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

Usage

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

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

The **ldap_first_attribute** and **ldap_next_attribute** APIs are used to step through the attributes in an LDAP entry. The **ldap_first_attribute** API takes an *entry* as returned by **ldap_first_entry** or

ldap_next_entry and returns a pointer to a buffer containing the name of the first attribute type in the entry. This buffer must be deallocated when its use is completed using **ldap_memfree**.

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

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

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

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

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

Error Conditions

If an error occurs for **ldap_first_attribute** and **ldap_next_attribute**, **NULL** is returned. Use **ldap_get_errno** to retrieve the error value. See “ldap_error” on page 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_get_values

ldap_memfree

ldap_error

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

<i>ld</i>	Specifies the LDAP handle returned by a previous call to ldap_open , ldap_ssl_init , or ldap_init .
<i>entry</i>	Specifies a pointer to an entry returned on a previous call to ldap_first_entry or ldap_next_entry .
<i>result</i>	Specifies the result as returned by a call to ldap_result or to one the synchronous LDAP search routines (see “ldap_search” on page 83).
<i>serverctrlsp</i>	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 .
<i>ref</i>	Specifies a pointer to a search continuation reference returned on a previous call to ldap_first_reference or ldap_next_reference .
<i>referralsp</i>	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.
<i>freeit</i>	Specifies a boolean value that determines if the LDAP result chain (as specified by <i>ref</i>) is to be freed. Any nonzero value will result in the LDAP result chain being freed after the requested information is extracted. Alternatively, the ldap_msgfree API can be used to free the LDAP result chain at a later time.

Usage

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

Processing Entries

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

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

ldap_first_entry/reference

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** and **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_search

ldap_first_attribute
ldap_get_values

ldap_get_dn

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

<i>ld</i>	Specifies the LDAP handle returned by a previous call to ldap_open , ldap_ssl_init , or ldap_init .
<i>entry</i>	Specifies attribute information as returned by ldap_first_entry or ldap_next_entry .
<i>dn</i>	Specifies the distinguished name of the entry to be parsed.
<i>notypes</i>	Requests that only the relative distinguished name (RDN) values be returned, not their types. For example, the DN <code>cn=Bob, c=US</code> would return as either <code>{“cn=Bob”, “c=US”, NULL}</code> or <code>{“Bob”, “US”, NULL}</code> depending on whether <i>notypes</i> 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_open**, **ldap_ssl_init**, or **ldap_init**.

<i>entry</i>	Specifies the LDAP entry from which to retrieve the attribute values.
<i>attr</i>	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.
<i>vals</i>	Specifies a pointer to a NULL-terminated array of attribute values returned by ldap_get_values .
<i>bvals</i>	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:

ldap_get_values

```
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_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.

Format

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

```
LDAP *ldap_init(
    char *host,
    int port)
```

```
LDAP *ldap_open(
    char *host,
    int port)
```

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

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

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

Parameters

Input

host

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

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

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

ldap_init

<i>port</i>	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 .
<i>ld</i>	Specifies the LDAP handle returned by a previous call to ldap_open , ldap_ssl_init , or ldap_init .
<i>optionToSet</i>	Specifies which LDAP option's value should be set. See “Setting and Getting Session Settings” on page 59 for the list of supported options.
<i>optionToGet</i>	Specifies which LDAP option's value should be returned. See “Setting and Getting Session Settings” on page 59 for the list of supported options.
<i>optionValue</i>	Depending on the operation, protocol version, or both, <i>optionValue</i> 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 <i>optionValue</i> parameter to be specified.

Table 3. The optionValue Parameter Specifications

<i>optionToSet</i> or <i>optionToGet</i>	ldap_set_option (Version 3)	ldap_set_option (Version 2)	ldap_set_option_np	ldap_get_option
LDAP_OPT_SIZELIMIT	int *	int	int	int *
LDAP_OPT_TIMELIMIT	int *	int	int	int *
LDAP_OPT_REFHOPLIMIT	int *	int	int	int *
LDAP_OPT_DEREF	int *	int	int	int *
LDAP_OPT_RESTART	int (ON/OFF)	int (ON/OFF)	int (ON/OFF)	int *
LDAP_OPT_REFFERALS	int (ON/OFF)	int (ON/OFF)	int (ON/OFF)	int *
LDAP_OPT_DEBUG	int *	int	int	int *
LDAP_OPT_SSL_CIPHER	char *	char *	char *	char **
LDAP_OPT_SSL_TIMEOUT	int *	int	int	int *
LDAP_OPT_REBIND_FN	LDAPRebindProc *	LDAPRebindProc *	LDAPRebindProc *	LDAPRebindProc **
LDAP_OPT_PROTOCOL_VERSION	int *	int *	int	int *
LDAP_OPT_SERVER_CONTROLS	LDAPControl **	n/a	LDAPControl **	LDAPControl ***
LDAP_OPT_CLIENT_CONTROLS	LDAPControl **	n/a	LDAPControl **	LDAPControl ***
LDAP_OPT_UTF8_IO	int (ON/OFF)	int (ON/OFF)	int (ON/OFF)	int *
LDAP_OPT_V2_WIRE_FORMAT	int	int	int	int *
LDAP_OPT_HOST_NAME	n/a	n/a	n/a	char *
LDAP_OPT_ERROR_NUMBER	n/a	n/a	n/a	int *
LDAP_OPT_ERROR_STRING	n/a	n/a	n/a	char *
LDAP_OPT_EXT_ERROR	n/a	n/a	n/a	int *

Note: The ON and OFF in the table refer to **LDAP_OPT_ON** and **LDAP_OPT_OFF**, respectively.

Usage

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

For SSL, the equivalent of **ldap_init** is **ldap_ssl_init**. The **ldap_ssl_init** API is used to initialize a secure SSL session with a server. See “**ldap_ssl**” on page 88 for more information.

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

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

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

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

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 13. Setting the **LDAP_DEBUG** environment variable has the same effect as calling **ldap_set_option** to set the **LDAP_OPT_DEBUG** session option.

The **LDAP_VERSION** environment variable can be used to establish the LDAP version to be used for a session. Setting the **LDAP_VERSION** environment variable has the same effect as calling **ldap_set_option** to set the **LDAP_OPT_PROTOCOL_VERSION** session option. Valid values for the **LDAP_VERSION** environment variable are **2** and **3**. See “**LDAP_OPT_PROTOCOL_VERSION**” on page 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 66 for more information.

Setting and Getting Session Settings

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

- Get or set the maximum number of entries that can be returned on a search operation. (**LDAP_OPT_SIZELIMIT**)
- Get or set the maximum number of seconds to wait for search results. (**LDAP_OPT_TIMELIMIT**)
- Get or set the maximum number of referrals in a sequence that the client can follow. (**LDAP_OPT_REFHOPLIMIT**)
- Get or set the rules for following aliases at the server. (**LDAP_OPT_DEREF**)
- Get or set whether select system call should be restarted. (**LDAP_OPT_RESTART**)
- Get or set whether referrals should be followed by the client. (**LDAP_OPT_REFERRALS**)
- Get or set the debug options. (**LDAP_OPT_DEBUG**)
- Get or set the SSL ciphers to use. (**LDAP_OPT_SSL_CIPHER**)
- Get or set the SSL timeout for refreshing session keys. (**LDAP_OPT_SSL_TIMEOUT**)
- Get or set the address of application's rebind procedure. (**LDAP_OPT_REBIND_FN**)

ldap_init

- Get or set the LDAP protocol version to use (Version 2 or Version 3). (**LDAP_OPT_PROTOCOL_VERSION**)
- Get or set the default server controls. (**LDAP_OPT_SERVER_CONTROLS**)
- Get or set the default client library controls. (**LDAP_OPT_CLIENT_CONTROLS**)
- Get or set the format of textual data. (**LDAP_OPT_UTF8_IO**)
- Get or set the format of textual data when using V2 protocol. (**LDAP_OPT_V2_WIRE_FORMAT**)
- Get the current host name (cannot be set). (**LDAP_OPT_HOST_NAME**)
- 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 67 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_REFHOPLIMIT

| 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:

ldap_init

```
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_REFERRALS, (void *) LDAP_OPT_ON); /* Version 2 or 3 protocol */
    or
ldap_set_option_np( ld, LDAP_OPT_REFERRALS, (int) LDAP_OPT_ON);

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

LDAP_OPT_DEBUG

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

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

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

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

Examples:

```

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

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

```

Example turning all traces on:

```

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

```

Example turning all tracing off:

```

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

```

Example tracing just BER encodings and functional flow tracepoints:

```

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

```

Example tracing packets and connections:

```

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

```

LDAP_OPT_SSL_CIPHER

Specifies a set of one or more ciphers to be used when negotiating the cipher algorithm with the LDAP Server. The value for this option is specified as the *v3cipher_specs* value supplied to the **gsk_secure_soc_init** function call in System SSL. Refer to the *OS/390 Cryptographic Services System Secure Sockets Layer Programming Guide and Reference* 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_init

- **LDAP_SSL_RC4_MD5_US** "04"
- **LDAP_SSL_DES_SHA_US** "09"
- **LDAP_SSL_3DES_SHA_US** "0A"

Examples:

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

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

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

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

LDAP_OPT_SSL_TIMEOUT

Specifies in seconds the SSL inactivity timer. After the specified seconds, in which no SSL activity has occurred, the SSL connection will be refreshed with new session keys. A smaller value may help increase security, but will have an impact on performance. The default SSL timeout 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_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 OS/390 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 OS/390 LDAP Version 3 servers. Also note that many non-OS/390 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 the OS/390 Release 8 version of the LDAP client library.

Examples:

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

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

LDAP_OPT_HOST_NAME

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

Example:

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

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

LDAP_OPT_ERROR_NUMBER

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

Example:

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

LDAP_OPT_ERROR_STRING

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

Example:

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

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

LDAP_OPT_EXT_ERROR

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

Example:

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

Returns errors reported by the SSL library.

Error Conditions

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

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

ldap_set_option Syntax for LDAP Version 2 Applications

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

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

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

Comparing the **ldap_set_option** and **ldap_set_option_np** APIs

The **ldap_set_option** and **ldap_set_option_np** APIs support the same LDAP option value settings; they differ only in the level of indirection required to specify certain settings. The **ldap_set_option_np** API is a 390-specific API and its intent is to provide an alternate programming interface for setting LDAP option values. Furthermore, the rules for specifying values through **ldap_set_option_np** will not be subject to change in future releases. Unlike **ldap_set_option**, the **ldap_set_option_np** API expects the *value* of the following option values to be supplied, instead of the address of the value, regardless of the LDAP version setting:

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

Related Topics

ldap_bind

ldap_memfree

ldap_memfree
ldap_control_free
ldap_controls_free

Purpose

Free storage allocated by the LDAP library.

Format

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

```
void ldap_memfree(  
    char *mem)
```

```
void ldap_control_free(  
    LDAPControl *ctrl)
```

```
void ldap_controls_free(  
    LDAPControl **ctrls)
```

Parameters

Input

<i>mem</i>	Specifies the pointer to a character string that was previously allocated by the LDAP client library and is no longer needed by the application.
<i>ctrl</i>	Specifies the address of an LDAPControl structure.
<i>ctrls</i>	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

<i>ld</i>	Specifies the LDAP handle returned by a previous call to ldap_open , ldap_ssl_init , or ldap_init .
<i>result</i>	Specifies the result returned by a call to ldap_result or one of the synchronous search routines (see “ldap_search” on page 83).
<i>msg</i>	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

<i>ld</i>	Specifies the LDAP handle returned by a previous call to ldap_open , ldap_ssl_init , or ldap_init .
<i>dn</i>	Specifies the distinguished name (DN) of the entry to be modified.
<i>mods</i>	<p>A NULL-terminated array of modifications to make to the entry. Each element of the <i>mods</i> array is a pointer to an LDAPMod structure.</p> <p>The <i>mod_op</i> 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 <i>mod_type</i> and <i>mod_values</i> fields specify the attribute type to modify and a NULL-terminated array of values to add, delete, or replace respectively. The <i>mod_next</i> field is used only by the LDAP library and should be ignored by the client.</p> <p>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 <i>mod_op</i> to the logical OR of the operation as above (for example, LDAP_MOD_REPLACE) and the constant LDAP_MOD_BVALUES. In this case, <i>mod_bvalues</i> should be used instead of <i>mod_values</i>, 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.</p> <p>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 <i>mod_values</i> 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.</p>
<i>freemods</i>	Specifies whether to deallocate the <i>mods</i> pointer. If <i>freemods</i> is nonzero, the <i>mods</i> pointer itself is deallocated as well.
<i>serverctrls</i>	Specifies a list of LDAP server controls. This parameter may be set to NULL. See “LDAP Controls” on page 25 for more information about server controls.
<i>clientctrls</i>	Specifies a list of LDAP client controls. This parameter may be set to NULL. See “LDAP Controls” on page 25 for more information about client controls.

Output

<i>msgidp</i>	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

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 **matchddnp,
    char **errmsgp,
    char ***referralsp,
    LDAPControl ***servctrlsp,
    int freeit)

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

Parameters

Input

<i>ld</i>	Specifies the LDAP handle returned by a previous call to ldap_open , ldap_ssl_init , or ldap_init .
<i>res</i>	Specifies the result of an LDAP operation as returned by ldap_result or one of the synchronous LDAP API operation calls.
<i>errcodep</i>	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 <i>errcodep</i> if the LDAPResult message is to be ignored.
<i>matchddnp</i>	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 <i>matchddnp</i> if the matched DN is to be ignored. The matched DN string should be freed by calling ldap_memfree .
<i>errmsgp</i>	Specifies a pointer to a result parameter that is filled in with the contents of the referrals field from the LDAPMessage message. The error message string should be freed by calling ldap_memfree .

ldap_parse_result

<i>referralsp</i>	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.
<i>serverctrlsp</i>	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 .
<i>freeit</i>	Specifies a boolean value that determines if the LDAP result (as specified by <i>res</i>) is to be freed. Any nonzero value will result in <i>res</i> being freed after the requested information is extracted. Alternatively, the ldap_msgfree API can be used to free the result at a later time.
<i>servercredp</i>	Specifies a pointer to a result parameter. For SASL bind results, this result parameter will be filled in with the credentials returned by the server for mutual authentication (if returned). The credentials, if returned, are returned in a berval structure. NULL may be supplied to ignore this field.

Usage

The **ldap_parse_result** API is used to:

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

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

Error Conditions

The parse APIs return an LDAP error code if they encounter an error parsing the result. See “**ldap_error**” on page 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_open**, **ldap_ssl_init**, or **ldap_init**.

ldap_rename

<i>dn</i>	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, <i>dn</i> specifies the distinguished name (DN) of the entry whose relative distinguished name (RDN) is to be changed.
<i>newrdn</i>	Specifies the new RDN to give the entry.
<i>newparent</i>	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 <i>newparent</i> parameter should always be NULL when using Version 2 of the LDAP protocol; otherwise the server's behavior is undefined.
<i>deleteoldrdn</i>	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 <i>newrdn</i> is different from the old RDN.
<i>serverctrls</i>	Specifies a list of LDAP server controls. This parameter may be set to NULL. See "LDAP Controls" on page 25 for more information about server controls.
<i>clientctrls</i>	Specifies a list of LDAP client controls. This parameter may be set to NULL. See "LDAP Controls" on page 25 for more information about client controls.

Output

<i>msgidp</i>	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

<i>ld</i>	Specifies the LDAP handle returned by a previous call to ldap_open , ldap_ssl_init , or ldap_init .
<i>msgid</i>	Contains an invocation identifier returned when an operation was initiated. Provide the <i>msgid</i> if the result of a specific operation is required, otherwise supply LDAP_RES_ANY .
<i>all</i>	<p>For search responses, selects whether a single entry of the search should be returned or all results of the search should be returned.</p> <p>A search response is made up of zero or more search entries followed by a search result. If <i>all</i> is set to 0, search entries will be returned one at a time as they come in, through separate calls to ldap_result. If <i>all</i> is nonzero, the search response will only be returned in its entirety, that is, after all entries and the final search result have been received. Specify LDAP_MSG_RECEIVED to indicate that all results retrieved so far should be returned in the result chain.</p>

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 83).

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 48.

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

LDAP_RES_BIND
LDAP_RES_SEARCH_ENTRY
LDAP_RES_SEARCH_RESULT
LDAP_RES_MODIFY
LDAP_RES_ADD
LDAP_RES_DELETE
LDAP_RES_MODRDN
LDAP_RES_COMPARE
LDAP_RES_SEARCH_REFERENCE
LDAP_RES_EXTENDED
LDAP_RES_ANY

Usage

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

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

The **ldap_msgfree** API is used to deallocate the memory allocated for a result by **ldap_result** or the synchronous LDAP search operation routines (for example, **ldap_search_s** and **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,

```

ldap_search

```
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

<i>ld</i>	Specifies the LDAP handle returned by a previous call to ldap_open , ldap_ssl_init , or ldap_init .
<i>base</i>	<p>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 <i>A String Representation of Distinguished Names</i>. Following is an example:</p> <p>cn=Jane Doe, o=Your Company, c=US</p>
<i>scope</i>	<p>Specifies the scope of the search and must be one of the following:</p> <ul style="list-style-type: none">LDAP_SCOPE_BASE to search the entry named by base itselfLDAP_SCOPE_ONELEVEL to search the entry's immediate childrenLDAP_SCOPE_SUBTREE to search the entry and all its descendents
<i>filter</i>	<p>A string representation of the filter to apply in the search. Simple filters can be specified as <i>attributetype=attributevalue</i>. More complex filters are specified using a prefix notation according to the following BNF:</p> <pre><filter> ::= '(' <filtercomp> ')' <filtercomp> ::= <and> ; <or> ; <not> ; <simple> <and> ::= '&' <filterlist> <or> ::= ';' <filterlist> <not> ::= '!' <filter> <filterlist> ::= <filter> ; <filter> <filterlist> <simple> ::= <attributetype> <filtertype> <attributevalue> <filtertype> ::= '=' ; '~=' ; '<=' ; '>='</pre> <p>The “~=” construct is used to specify approximate matching. The representation for <i><attributetype></i> and <i><attributevalue></i> are as described in IETF RFC 1778, <i>The String Representation of Standard Attribute Syntaxes</i>. In addition, <i><attributevalue></i> can be a single asterisk (*) to achieve an attribute existence test, or can contain text and asterisks (*) interspersed to achieve substring matching.</p> <p>For example, the filter</p> <pre>mail=*</pre> <p>finds any entries that have a mail attribute. The filter</p> <pre>mail=*@student.of.life.edu</pre> <p>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</p>

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))
```

<i>attrs</i>	Specifies a NULL-terminated array of character string attribute types to return from entries that match <i>filter</i> . If NULL is specified, all attributes are returned.
<i>attrsonly</i>	Specifies attribute information. If nonzero, only attribute types are returned. If zero, both attribute types and attribute values are returned.
<i>timeout</i>	Specifies blocking for ldap_search_st . If <i>timeout</i> is not NULL , it specifies a maximum interval to wait for the selection to complete. If <i>timeout</i> is NULL , the select blocks indefinitely until the result for the operation identified by the <i>msgid</i> is received. To poll, the <i>timeout</i> 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.
<i>serverctrls</i>	Specifies a list of LDAP server controls. This parameter may be set to NULL . See “LDAP Controls” on page 25 for more information about server controls.
<i>clientctrls</i>	Specifies a list of LDAP client controls. This parameter may be set to NULL . See “LDAP Controls” on page 25 for more information about client controls.
<i>sizelimit</i>	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

<i>res</i>	Specifies the result of an LDAP operation as returned by ldap_result or one of the synchronous LDAP API operation calls.
<i>msgidp</i>	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).

ldap_search

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.

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

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

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

LDAP_OPT_SIZELIMIT	A limit on the number of entries to return from the search. A value of zero means no limit. Note that the value from the session handle is ignored when using the ldap_search_ext or ldap_search_ext_s functions.
LDAP_OPT_TIMELIMIT	A limit on the number of seconds to spend on the search. A value of zero means no limit. Note that the value from the session handle is ignored when using the ldap_search_ext or ldap_search_ext_s APIs.
LDAP_OPT_DEREF	One of LDAP_DEREF_NEVER (0x00), LDAP_DEREF_SEARCHING (0x01), LDAP_DEREF_FINDING (0x02), or LDAP_DEREF_ALWAYS (0x03), specifying how aliases should be handled during the search. The LDAP_DEREF_SEARCHING value means aliases should be dereferenced during the search but not when locating the base object of the search. The LDAP_DEREF_FINDING value means aliases should be dereferenced when locating the base object but not during the search.

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

Reading an Entry

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

Listing the Children of an Entry

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

Error Conditions

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

[ldap_ssl_client_init](#)
[ldap_ssl_init](#)
[ldap_ssl_start](#) (deprecated)

Purpose

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

Format

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

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

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

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

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

Parameters

Input

<i>ld</i>	Specifies the LDAP pointer returned by a previous call to ldap_ssl_init , ldap_init or ldap_open .
<i>host</i>	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 <i>host:port</i> . If present, the <i>:port</i> overrides the ldap_ssl_init <i>port</i> parameter. If the <i>host</i> parameter is NULL, the LDAP Server will be assumed to be running on the local host.
<i>port</i>	Specifies the port number to which to connect. If the default IANA-assigned SSL port of 636 is desired, LDAPS_PORT should be specified.
<i>keyring</i>	Specifies the name of a key database file. The key database file typically contains one or more certificates of certificate authorities (CAs) that are trusted by the client. These types of X.509 certificates are also known as <i>trusted roots</i> . A key database file can also be used to store the client's private key or keys and associated client certificate or certificates. A private

key and associated client certificate are required only if the LDAP Server is configured to require client and server authentication. If the LDAP Server is configured to provide only server authentication, a private key and client certificate are not required.

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

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

For more information on using GSKKMAN to manage the contents of a key database file, see the *OS/390 Security Server LDAP Server Administration and Usage Guide* and the *OS/390 Cryptographic Services System Secure Sockets Layer Programming Guide and Reference*.

keyring_pw

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

name

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

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

ssl_timeout

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

pSSLReasonCode

Specifies a pointer to the SSL Reason Code, which provides additional information in the event that an error occurs during initialization of the SSL stack (when **ldap_ssl_client_init** is invoked). See "ldapssl.h" on page 105 for reason codes that can be returned.

Usage

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

ldap_ssl

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

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

The LDAP session handle returned by **ldap_ssl_init** (and **ldap_init**) is a pointer to an opaque data type representing an LDAP session. The **ldap_get_option** and **ldap_set_option** APIs are used to access and set a variety of session-wide parameters. See “ldap_init” on page 57 for more information about **ldap_get_option** and **ldap_set_option**.

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

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

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

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

... additional LDAP API calls

```
rc = ldap_unbind( ld );
```

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

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

```
RC4_MD5_EXPORT
RC2_MD5_EXPORT
RC4_SHA_US
RC4_MD5_US
DES_SHA_US
3DES_SHA_US
```

See “ldap_init” on page 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 session-level timer, use:

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

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

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

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

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

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

For more information on **ldap_set_option**, see "ldap_init" on page 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 versions of these utilities include RSA software.

Related Topics

ldap_init

ldap_url

```

    ldap_is_ldap_url
    ldap_url_parse
    ldap_free_urldesc
    ldap_url_search
    ldap_url_search_s
    ldap_url_search_st

```

Purpose

LDAP Uniform Resource Locator (URL) routines.

Format

```

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

#include <ldap.h>

int ldap_is_ldap_url(
    char *url)

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

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

void ldap_free_urldesc(
    LDAPURLDesc *ludp)

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

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

```

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

Parameters

Input

<i>ld</i>	Specifies the LDAP handle returned by a previous call to ldap_open , ldap_ssl_init , or ldap_init .
<i>url</i>	Specifies the LDAP URL.
<i>ldup</i>	Specifies the URL description.
<i>attrsonly</i>	Specifies attribute information. Set to 1 to request attributes types only. Set to 0 to request both attribute types and attribute values.
<i>timeout</i>	Specifies blocking for ldap_search_st . If <i>timeout</i> is not NULL , it specifies a maximum interval to wait for the selection to complete. If <i>timeout</i> is NULL , the select blocks indefinitely until the result for the operation identified by the <i>msgid</i> is received. To poll, the <i>timeout</i> parameter should be non-null, pointing to a zero-valued timeval structure.
<i>ldupp</i>	Points to the LDAP URL description, as returned by ldap_url_parse .

Output

<i>ldupp</i>	Points to the LDAP URL description, as returned by ldap_url_parse .
<i>res</i>	On successful completion of the search, <i>res</i> is set to point to a set of LDAPMessage structures. These should be parsed with ldap_first_entry and ldap_next_entry .

Usage

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

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

where:

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

For example,

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

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

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

ldap_url

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

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

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

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

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

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

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

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

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

Notes

For search operations, if *hostport* is omitted, host and port for the current connection are used. If *hostport* is specified, and is different from the host and port combination used for the current connection, the search is directed to *hostport*, instead of using the current connection. In this case, the underlying referral mechanism is used to bind to *hostport*.

If the LDAP URL does not contain a search filter, the filter defaults to `objectClass=*`.

Error Conditions

If an error occurs for **ldap_url_parse**, one of the following values is returned:

LDAP_URL_ERR_NOTLDAP URL doesn't begin with ldap://

LDAP_URL_ERR_NODN URL has no DN (required)

LDAP_URL_ERR_BADSCOPE URL scope string is invalid

LDAP_URL_ERR_MEM can't allocate memory space

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

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.

The following shows the contents of the `lber.h` header file:

```
| ??=ifdef __COMPILER_VER__
| ??=pragma filetag ("IBM-1047")
| ??=endif
|
| /* @(#)36      1.4.4.3  src/libraries/libcommon/ber/lber.h, ldap.lib, os390r8ldap 4/11/99 13:13:04 */
|
| /*
|  * Licensed Materials - Property of IBM
|  * 5647-A01
|  * (C) Copyright IBM Corp. 1997, 1999
|  *
|  */
|
| /*
|  * Copyright (c) 1990 Regents of the University of Michigan.
|  * All rights reserved.
|  *
|  * Redistribution and use in source and binary forms are permitted
|  * provided that this notice is preserved and that due credit is given
|  * to the University of Michigan at Ann Arbor. The name of the
|  * University may not be used to endorse or promote products
|  * derived from this software without specific prior written
|  * permission. This software is provided "as is" without express
|  * or implied warranty.
|  */
|
| #ifndef _LBER_H
| #define _LBER_H
|
| /* structure for returning a sequence of octet strings + length */
| struct berval {
|     unsigned long    bv_len;
|     char             *bv_val;
| };
|
| typedef struct berelement BerElement;
| #define NULLBER      ((BerElement *) 0)
|
| #endif /* _LBER_H */
```

ldap.h

The **ldap.h** header file contains definitions for the LDAP routines. It has the following format:

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

It is a mandatory include file for all applications working with the LDAP APIs. This header defines constants, types, and macros that are used with the interface.

The following shows the contents of the **ldap.h** header file:

```
| ??=ifdef __COMPILER_VER__
| ??=pragma filetag ("IBM-1047")
| ??=endif

| /* %Z%M%          %I% %W% %G% %U% */
| /*
|  * Licensed Materials - Property of IBM
|  * 5647-A01
|  * (C) Copyright IBM Corp. 1997, 1999
|  *
|  */

| /*
|  * Copyright (c) 1990 Regents of the University of Michigan.
|  * All rights reserved.
|  *
|  * Redistribution and use in source and binary forms are permitted
|  * provided that this notice is preserved and that due credit is given
|  * to the University of Michigan at Ann Arbor. The name of the
|  * University may not be used to endorse or promote products
|  * derived from this software without specific prior written
|  * permission. This software is provided "as is" without express
|  * or implied warranty.
|  */

| #ifndef _LDAP_H
| #define _LDAP_H

| #ifdef __cplusplus
| extern "C" {
| #endif

| #ifdef _WIN32
| #include <winsock.h>
| #else
| #include <sys/time.h>
| #endif

| #include <lber.h>

| #define LDAP_VERSION2 2
| #define LDAP_VERSION3 3
| #ifdef LDAPV3
| #define LDAP_VERSION LDAP_VERSION3
| #else
| #define LDAP_VERSION LDAP_VERSION2
| #endif
```

```

| /* For compatibility w/Netscape implementation of ldap_version(). */
| #define LDAP_SECURITY_NONE      0

| #define LDAP_PORT      389
| #define LDAPS_PORT     636

| #define LDAP_MAX_ATTR_LEN  100

| /* possible result types a server can return */
| #define LDAP_RES_BIND      0x61L /* application + constructed */
| #define LDAP_RES_SEARCH_ENTRY 0x64L /* application + constructed */
| #define LDAP_RES_SEARCH_RESULT 0x65L /* application + constructed */
| #define LDAP_RES_MODIFY    0x67L /* application + constructed */
| #define LDAP_RES_ADD       0x69L /* application + constructed */
| #define LDAP_RES_DELETE    0x6bL /* application + constructed */
| #define LDAP_RES_MODRDN    0x6dL /* application + constructed */
| #define LDAP_RES_COMPARE   0x6fL /* application + constructed */
| #define LDAP_RES_SEARCH_REFERENCE 0x73L /* application + constructed */
| #define LDAP_RES_EXTENDED  0x78L /* application + constructed */
| #define LDAP_EXTENDED_RES_NAME 0x8aL /* context specific+primitive*/
| #define LDAP_EXTENDED_RES_VALUE 0x8bL /* context specific+primitive*/
| #define LDAP_RES_REFERRAL   0xa3L /* context specific+constructed*/
| #define LDAP_RES_ANY       (-1L)

| /* authentication methods available */
| #define LDAP_AUTH_SIMPLE    0x80L /* context specific + primitive */
| #define LDAP_AUTH_SASL_30   0xa3L
| #define LDAP_AUTH_SASL_SIMPLE ""

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

| /* bind constants */
| #define LDAP_MECHANISM_EXTERNAL "EXTERNAL"

| /* for modifications */
| typedef struct ldapmod {
|     int    mod_op;
| #define LDAP_MOD_ADD      0x00
| #define LDAP_MOD_DELETE  0x01
| #define LDAP_MOD_REPLACE 0x02
| #define LDAP_MOD_BVALUES 0x80
|     char    *mod_type;
|     union {
|         char    **modv_strvals;
|         struct berval **modv_bvals;
|     } mod_vals;
| #define mod_values    mod_vals.modv_strvals
| #define mod_bvalues   mod_vals.modv_bvals
|     struct ldapmod    *mod_next;
| } LDAPMod;

| /*
|  * options that can be set/gotten

```

```

|  */
| #define LDAP_OPT_SIZELIMIT      0x00
| #define LDAP_OPT_TIMELIMIT     0x01
| #define LDAP_OPT_REFERRALS     0x00000002
| #define LDAP_OPT_DEREF         0x03
| #define LDAP_OPT_RESTART       0x00000004
| #define LDAP_OPT_REFHOPLIMIT   0x05
| #define LDAP_OPT_DEBUG         0x06

| #define LDAP_OPT_SSL_CIPHER    0x07
| #define LDAP_OPT_SSL_TIMEOUT   0x08

| #define LDAP_OPT_REBIND_FN     0x09
| #define LDAP_OPT_SSL           0x0A
| #define LDAP_OPT_PROTOCOL_VERSION 0x11
| #define LDAP_OPT_SERVER_CONTROLS 0x12
| #define LDAP_OPT_CLIENT_CONTROLS 0x13
| #define LDAP_OPT_HOST_NAME     0x30
| #define LDAP_OPT_ERROR_NUMBER  0x31
| #define LDAP_OPT_ERROR_STRING  0x32
| #define LDAP_OPT_EXT_ERROR     0x33

| #define LDAP_OPT_UTF8_IO       0xE0
| #define LDAP_OPT_SSL_CERTIFICATE_DN 0xE1

| #define LDAP_OPT_V2_WIRE_FORMAT 0xE2

| #define LDAP_OPT_V2_WIRE_FORMAT_IS08859_1 0x00
| #define LDAP_OPT_V2_WIRE_FORMAT_UTF8 0x01

| #define LDAP_OPT_LCS          0x0F

| /* option value for no size limit or no time limit on searches */
| #define LDAP_NO_LIMIT        0

| /* option values for binary options */
| #define LDAP_OPT_ON          0x01
| #define LDAP_OPT_OFF         0x00

| /* option values for dereferencing aliases */
| #define LDAP_DEREF_NEVER     0
| #define LDAP_DEREF_SEARCHING 1
| #define LDAP_DEREF_FINDING  2
| #define LDAP_DEREF_ALWAYS   3

| /* default limit on nesting of referrals */
| #define LDAP_DEFAULT_REFHOPLIMIT 5

| /* 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_ANY       0xffffffff

| /* options for SSL ciphers */
| #define LDAP_SSL_RC4_MD5_EX  "03"
| #define LDAP_SSL_RC2_MD5_EX  "06"
| #define LDAP_SSL_RC4_SHA_US  "05"
| #define LDAP_SSL_RC4_MD5_US  "04"
| #define LDAP_SSL_DES_SHA_US  "09"
| #define LDAP_SSL_3DES_SHA_US "0A"

| /*
|  * possible error codes we can return
|  */

| #define LDAP_SUCCESS          0x00
| #define LDAP_OPERATIONS_ERROR 0x01
| #define LDAP_PROTOCOL_ERROR   0x02
| #define LDAP_TIMELIMIT_EXCEEDED 0x03
| #define LDAP_SIZELIMIT_EXCEEDED 0x04
| #define LDAP_COMPARE_FALSE     0x05
| #define LDAP_COMPARE_TRUE      0x06
| #define LDAP_STRONG_AUTH_NOT_SUPPORTED 0x07
| #define LDAP_STRONG_AUTH_REQUIRED 0x08
| #define LDAP_PARTIAL_RESULTS   0x09

| #define LDAP_REFERRAL          0x0a
| #define LDAP_ADMIN_LIMIT_EXCEEDED 0x0b
| #define LDAP_UNAVAILABLE_CRITICAL_EXTENSION 0x0c
| #define LDAP_CONFIDENTIALITY_REQUIRED 0x0d
| #define LDAP_SASLBIND_IN_PROGRESS 0x0e

| #define LDAP_NO_SUCH_ATTRIBUTE 0x10
| #define LDAP_UNDEFINED_TYPE    0x11
| #define LDAP_INAPPROPRIATE_MATCHING 0x12
| #define LDAP_CONSTRAINT_VIOLATION 0x13
| #define LDAP_TYPE_OR_VALUE_EXISTS 0x14
| #define LDAP_INVALID_SYNTAX    0x15

| #define LDAP_NO_SUCH_OBJECT     0x20
| #define LDAP_ALIAS_PROBLEM      0x21
| #define LDAP_INVALID_DN_SYNTAX  0x22
| #define LDAP_IS_LEAF            0x23
| #define LDAP_ALIAS_DEREF_PROBLEM 0x24

```

```

| #define LDAP_INAPPROPRIATE_AUTH 0x30
| #define LDAP_INVALID_CREDENTIALS 0x31
| #define LDAP_INSUFFICIENT_ACCESS 0x32
| #define LDAP_BUSY 0x33
| #define LDAP_UNAVAILABLE 0x34
| #define LDAP_UNWILLING_TO_PERFORM 0x35
| #define LDAP_LOOP_DETECT 0x36

| #define LDAP_NAMING_VIOLATION 0x40
| #define LDAP_OBJECT_CLASS_VIOLATION 0x41
| #define LDAP_NOT_ALLOWED_ON_NONLEAF 0x42
| #define LDAP_NOT_ALLOWED_ON_RDN 0x43
| #define LDAP_ALREADY_EXISTS 0x44
| #define LDAP_NO_OBJECT_CLASS_MODS 0x45
| #define LDAP_RESULTS_TOO_LARGE 0x46

| #define LDAP_AFFECTS_MULTIPLE_DSAS 0x47

| #define LDAP_OTHER 0x50
| #define LDAP_SERVER_DOWN 0x51
| #define LDAP_LOCAL_ERROR 0x52
| #define LDAP_ENCODING_ERROR 0x53
| #define LDAP_DECODING_ERROR 0x54
| #define LDAP_TIMEOUT 0x55
| #define LDAP_AUTH_UNKNOWN 0x56
| #define LDAP_FILTER_ERROR 0x57
| #define LDAP_USER_CANCELLED 0x58
| #define LDAP_PARAM_ERROR 0x59
| #define LDAP_NO_MEMORY 0x5a
| #define LDAP_CONNECT_ERROR 0x5b
| #define LDAP_NOT_SUPPORTED 0x5c
| #define LDAP_CONTROL_NOT_FOUND 0x5d
| #define LDAP_NO_RESULTS_RETURNED 0x5e
| #define LDAP_MORE_RESULTS_TO_RETURN 0x5f

| #define LDAP_URL_ERR_NOTLDAP 0x60
| #define LDAP_URL_ERR_NODN 0x61
| #define LDAP_URL_ERR_BADSCOPE 0x62
| #define LDAP_URL_ERR_MEM 0x63

| #define LDAP_CLIENT_LOOP 0x64
| #define LDAP_REFERRAL_LIMIT_EXCEEDED 0x65

| #define LDAP_SSL_ALREADY_INITIALIZED 0x70
| #define LDAP_SSL_INITIALIZE_FAILED 0x71
| #define LDAP_SSL_CLIENT_INIT_NOT_CALLED 0x72
| #define LDAP_SSL_PARAM_ERROR 0x73
| #define LDAP_SSL_HANDSHAKE_FAILED 0x74
| #define LDAP_SSL_GET_CIPHER_FAILED 0x75
| #define LDAP_SSL_NOT_AVAILABLE 0x76

| #define LDAP_NO_EXPLICIT_OWNER 0x80
| #define LDAP_NO_EXPLICIT_ACL 0x81

| /*
|  * This structure represents both ldap messages and ldap responses.
|  * These are really the same, except in the case of search responses,
|  * where a response has multiple messages.

```

```

|  */
|
| typedef struct ldapmsg LDAPMessage;
| #define NULLMSG ((LDAPMessage *) NULL)
|
| /*
|  * structure representing an ldap connection
|  */
| typedef struct ldap LDAP;
|
| /*
|  * type for ldap_set_rebind_proc()
|  */
| typedef int (*LDAPRebindProc)( struct ldap *ld, char **dn,
|      char **passwd, int *authmethodp, int freeit );
|
| /*
|  * types for ldap URL handling
|  */
| typedef struct ldap_url_desc {
|     char *lud_host;
|     int   lud_port;
|     char *lud_dn;
|     char **lud_attrs;
|     int   lud_scope;
|     char *lud_filter;
|     char *lud_string; /* for internal use only */
| } LDAPURLDesc;
| #define NULLLDAPURLEDESC ((LDAPURLDesc *)NULL)
|
| typedef struct _LDAPVersion {
|     int sdk_version;
|     int protocol_version;
|     int SSL_version;
|     int security_level;
|     char ssl_max_cipher[ 65 ] ;
|     char ssl_min_cipher[ 65 ] ;
| } LDAPVersion;
|
| typedef struct _LDAPControl {
|     char *ldctl_oid;
|     struct berval ldctl_value;
|     int      ldctl_iscritical;
| } LDAPControl;
|
| /* Function prototypes */
| #ifndef _NO_PROTO
| #define LDAP_P(x) x
| #else
| #define LDAP_P(x) ()
| #endif
|
| int ldap_abandon ( LDAP *ld, int msgid );
| int ldap_abandon_ext ( LDAP *ld, int msgid,
|     LDAPControl **serverctrls,
|     LDAPControl **clientctrls );
| int ldap_add ( LDAP *ld, char *dn, LDAPMod **attrs );

```

```

| int ldap_add_s ( LDAP *ld, char *dn, LDAPMod **attrs );
| int ldap_add_ext ( LDAP *ld, char *dn, LDAPMod **attrs,
|     LDAPControl **serverctrls,
|     LDAPControl **clientctrls,
|     int *msgidp );
| int ldap_add_ext_s ( LDAP *ld, char *dn, LDAPMod **attrs,
|     LDAPControl **serverctrls,
|     LDAPControl **clientctrls );
| int ldap_bind ( LDAP *ld, char *who, char *passwd,
|     int authmethod );
| int ldap_bind_s ( LDAP *ld, char *who, char *cred,
|     int method );
| int ldap_simple_bind ( LDAP *ld, char *who, char *passwd );
| int ldap_simple_bind_s ( LDAP *ld, char *who, char *passwd );
| void ldap_set_rebind_proc ( LDAP *ld,
|     LDAPRebindProc rebindproc );
| int ldap_compare ( LDAP *ld, char *dn, char *attr,
|     char *value );
| int ldap_compare_s ( LDAP *ld, char *dn, char *attr,
|     char *value );
| int ldap_compare_ext ( LDAP *ld, char *dn, char *attr,
|     struct berval *bvalue,
|     LDAPControl **serverctrls,
|     LDAPControl **clientctrls,
|     int *msgidp );
| int ldap_compare_ext_s ( LDAP *ld, char *dn, char *attr,
|     struct berval *bvalue,
|     LDAPControl **serverctrls,
|     LDAPControl **clientctrls );
| int ldap_delete ( LDAP *ld, char *dn );
| int ldap_delete_s ( LDAP *ld, char *dn );
| int ldap_delete_ext ( LDAP *ld, char *dn,
|     LDAPControl **serverctrls,
|     LDAPControl **clientctrls,
|     int *msgidp );
| int ldap_delete_ext_s ( LDAP *ld, char *dn,
|     LDAPControl **serverctrls,
|     LDAPControl **clientctrls );
| int ldap_result2error ( LDAP *ld, LDAPMessage *r, int freeit );
| char *ldap_err2string ( int err );
| void ldap_perror ( LDAP *ld, char *s );
| int ldap_get_errno ( LDAP *ld );
| int ldap_modify ( LDAP *ld, char *dn, LDAPMod **mods );
| int ldap_modify_s ( LDAP *ld, char *dn, LDAPMod **mods );
| int ldap_modify_ext ( LDAP *ld, char *dn, LDAPMod **mods,
|     LDAPControl **serverctrls,
|     LDAPControl **clientctrls,
|     int *msgidp );
| int ldap_modify_ext_s ( LDAP *ld, char *dn, LDAPMod **mods,
|     LDAPControl **serverctrls,
|     LDAPControl **clientctrls );
| 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 *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 *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 **matcheddn, char **errormsg,
| char ***referralsp, LDAPControl ***serverctrlsp,
| int freeint );
| int ldap_parse_sasl_bind_result ( LDAP* ld, LDAPMessage *result,
| struct berval **servercredp,
| int freeit );
| int ldap_parse_extended_result( LDAP *ld, LDAPMessage *res,
| char **resultoidp, struct berval **resultdata, int freeit);

| #ifdef __cplusplus
| }
| #endif

```

```
| #endif /* _LDAP_H */
```

ldapssl.h

The **ldapssl.h** header file contains definitions for the LDAP SSL routines. It is an include for all applications working with the LDAP SSL APIs. This header defines constants that are used with this interface. The following shows the contents of the **ldapssl.h** header file:

```
| ??=ifdef __COMPILER_VER__
| ??=pragma filetag ("IBM-1047")
| ??=endif

| /* @(#)43      1.3.3.9  ldapssl.h, ldap.libs, os390r8ldap 4/11/99 14:47:35 */

| /*
|  * Licensed Materials - Property of IBM
|  * 5647-A01
|  * (C) Copyright IBM Corp. 1997, 1999
|  *
|  */

| #ifndef _LDAPSSL_H
| #define _LDAPSSL_H

| /*
|  Return values returned from ldap_ssl_client_init(), ldap_ssl_init()
|  and ldap_ssl_start()
|  */

| #define LDAP_SSL_INITIALIZE_OK          0 /* Successful Completion      */
| #define LDAP_SSL_KEYFILE_IO_ERROR      1 /* Attention: Keyring io error */
| #define LDAP_SSL_KEYFILE_OPEN_FAILED   2 /* Attention: Keyring open error*/
| #define LDAP_SSL_KEYFILE_BAD_FORMAT    3 /* Attention: Keyring format bad*/
| #define LDAP_SSL_KEYFILE_BAD_PASSWORD  4 /* Attention: Keyring pw bad   */
| #define LDAP_SSL_KEYFILE_BAD_MALLOC    5 /* Error: Malloc failed        */
| #define LDAP_SSL_KEYFILE_NOTHING_TO_WRITE 6
| #define LDAP_SSL_KEYFILE_WRITE_FAILED  7
| #define LDAP_SSL_KEYFILE_NOT_FOUND     8
| #define LDAP_SSL_KEYFILE_BAD_DNAME     9 /* Error: Distinguished name bad*/
| #define LDAP_SSL_KEYFILE_BAD_KEY       10
| #define LDAP_SSL_KEYFILE_KEY_EXISTS    11
| #define LDAP_SSL_KEYFILE_BAD_LABEL     12
| #define LDAP_SSL_KEYFILE_DUPLICATE_NAME 13
| #define LDAP_SSL_KEYFILE_DUPLICATE_KEY  14
| #define LDAP_SSL_KEYFILE_DUPLICATE_LABEL 15
| #define LDAP_SSL_ERR_INIT_PARM_NOT_VALID 100 /* Error: Cipher spec bad     */
| #define LDAP_SSL_INIT_HARD_RT          101 /* Attention: No keyring file
|                                     or password */
| #define LDAP_SSL_INIT_SEC_TYPE_NOT_VALID 102 /* Error: Security type bad   */
| #define LDAP_SSL_INIT_V2_TIMEOUT_NOT_VALID 103 /* Error:V2 timeout value bad*/
| #define LDAP_SSL_INIT_V3_TIMEOUT_NOT_VALID 104 /* Error:V3 timeout value bad*/
| #define LDAP_SSL_KEYFILE_CERT_EXPIRED   105 /* Error:Certificate expired */

| /*
|  Return codes. These are returned as an LDAP_OPT_EXTERERROR, using
|  ldap_get_option(), when an SSL-related error has occurred.
```

```

|     Use ldap_get_option() with LDAP_OPT_EXTERERROR to get a more detailed SSL
|     error code whenever LDAP_SSL_HANDSHAKE_FAILED is returned from an
|     LDAP call
|     to use the new #defines listed above.
| */

| #define LDAP_SSL_SOC_BAD_V2_CIPHER                -40
| #define LDAP_SSL_SOC_BAD_V3_CIPHER                -41
| #define LDAP_SSL_SOC_BAD_SEC_TYPE                 -42
| #define LDAP_SSL_SOC_BAD_SEC_TYPE_COMBINATION     -102
| #define LDAP_SSL_SOC_NO_READ_FUNCTION             -43
| #define LDAP_SSL_SOC_NO_WRITE_FUNCTION            -44

| #define LDAP_SSL_ERROR_NO_CIPHERS                 -1
| #define LDAP_SSL_ERROR_NO_CERTIFICATE             -2
| #define LDAP_SSL_ERROR_BAD_CERTIFICATE            -4
| #define LDAP_SSL_ERROR_UNSUPPORTED_CERTIFICATE_TYPE -6
| #define LDAP_SSL_ERROR_IO                         -10
| #define LDAP_SSL_ERROR_BAD_MESSAGE               -11
| #define LDAP_SSL_ERROR_BAD_MAC                   -12
| #define LDAP_SSL_ERROR_UNSUPPORTED                -13
| #define LDAP_SSL_ERROR_BAD_CERT_SIG              -14
| #define LDAP_SSL_ERROR_BAD_CERT                  -15
| #define LDAP_SSL_ERROR_BAD_PEER                   -16
| #define LDAP_SSL_ERROR_PERMISSION_DENIED          -17
| #define LDAP_SSL_ERROR_SELF_SIGNED                -18
| #define LDAP_SSL_ERROR_BAD_MALLOC                 -20
| #define LDAP_SSL_ERROR_BAD_STATE                  -21    /* V3 */
| #define LDAP_SSL_ERROR_SOCKET_CLOSED              -22
| #define LDAP_SSL_ERROR_LDAP_SSL_INITIALIZATION_FAILED -23
| #define LDAP_SSL_ERROR_HANDLE_CREATION_FAILED     -24
| #define LDAP_SSL_ERROR_UNKNOWN_ERROR              -99

| /*
| -----
|     All #defines below this comment are slated for removal; all code should
|     be changed
|     to use the new #defines listed above.
| -----
| */

| /*
|     Return values returned from ldap_ssl_client_init(), ldap_ssl_init()
|     and ldap_ssl_start()
| */

| #define SKIT_INITIALIZE_OK                        0    /* Successful Completion */
| #define SKIT_KEYFILE_IO_ERROR                     1    /* Warning: keyring io error */
| #define SKIT_KEYFILE_OPEN_FAILED                  2    /* Warning: keyring open error */
| #define SKIT_KEYFILE_BAD_FORMAT                   3    /* Warning: keyring format bad */
| #define SKIT_KEYFILE_BAD_PASSWORD                 4    /* Warning: Keyring password bad */
| #define SKIT_KEYFILE_BAD_MALLOC                   5    /* Error: Malloc failed */
| #define SKIT_KEYFILE_NOTHING_TO_WRITE             6
| #define SKIT_KEYFILE_WRITE_FAILED                 7
| #define SKIT_KEYFILE_NOT_FOUND                    8
| #define SKIT_KEYFILE_BAD_DNAME                    9    /* Error: distinguish name bad */
| #define SKIT_KEYFILE_BAD_KEY                     10

```

```

| #define SKIT_KEYFILE_KEY_EXISTS      11
| #define SKIT_KEYFILE_BAD_LABEL      12
| #define SKIT_KEYFILE_DUPLICATE_NAME 13
| #define SKIT_KEYFILE_DUPLICATE_KEY  14
| #define SKIT_KEYFILE_DUPLICATE_LABEL 15
| #define SKIT_ERR_INIT_PARM_NOT_VALID 100 /* Error: cipher spec bad */
| #define SKIT_INIT_HARD_RT            101 /* Warning: no keyring file or pw*/
| #define SKIT_INIT_SEC_TYPE_NOT_VALID 102 /* Error: security type bad */
| #define SKIT_INIT_V2_TIMEOUT_NOT_VALID 103 /* Error: V2 timeout value bad */
| #define SKIT_INIT_V3_TIMEOUT_NOT_VALID 104 /* Error: V3 timeout value bad */
| #define SKIT_KEYFILE_CERT_EXPIRED    105 /* Error: Certificate expired */

| /*
|  Return codes. These are returned as an LDAP_OPT_EXTERERROR, using
|  ldap_get_option(), when an SSL-related error has occurred.

|  Use ldap_get_option() with LDAP_OPT_EXTERERROR to get a more detailed SSL
|  error code whenever LDAP_SSL_HANDSHAKE_FAILED is returned from an LDAP
|  call to use the new #defines listed above.
|  */

| #define SKIT_SOC_BAD_V2_CIPHER      -40
| #define SKIT_SOC_BAD_V3_CIPHER      -41
| #define SKIT_SOC_BAD_SEC_TYPE        -42
| #define SKIT_SOC_BAD_SEC_TYPE_COMBINATION -102
| #define SKIT_SOC_NO_READ_FUNCTION    -43
| #define SKIT_SOC_NO_WRITE_FUNCTION   -44

| #define SKIT_ERROR_NO_CIPHERS        -1
| #define SKIT_ERROR_NO_CERTIFICATE    -2
| #define SKIT_ERROR_BAD_CERTIFICATE   -4
| #define SKIT_ERROR_UNSUPPORTED_CERTIFICATE_TYPE -6
| #define SKIT_ERROR_IO                -10
| #define SKIT_ERROR_BAD_MESSAGE       -11
| #define SKIT_ERROR_BAD_MAC           -12
| #define SKIT_ERROR_UNSUPPORTED       -13
| #define SKIT_ERROR_BAD_CERT_SIG      -14
| #define SKIT_ERROR_BAD_CERT          -15
| #define SKIT_ERROR_BAD_PEER          -16
| #define SKIT_ERROR_PERMISSION_DENIED -17
| #define SKIT_ERROR_SELF_SIGNED        -18
| #define SKIT_ERROR_BAD_MALLOC         -20
| #define SKIT_ERROR_BAD_STATE          -21 /* V3 */
| #define SKIT_ERROR_SOCKET_CLOSED      -22
| #define SKIT_ERROR_SKIT_INITIALIZATION_FAILED -23
| #define SKIT_ERROR_HANDLE_CREATION_FAILED -24
| #define SKIT_ERROR_UNKNOWN_ERROR      -99

| #endif /* _LDAPSSL_H */

```

Appendix A. 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 10594-1785
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
522 South Road
Poughkeepsie, NY 12601-5400
U.S.A

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Programming Interface Information

This *OS/390 Security Server LDAP Client Application Development Guide and Reference* documents intended Programming Interfaces that allow the customer to write programs to obtain services of OS/390 LDAP.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States and/or other countries:

IBM	IBMLink	OS/390
AIX/6000	Library Reader	RACF
BookManager	OS/2	

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

Other company, product or service names may be the trademarks or service marks of others.

Glossary

This glossary defines new LDAP Server terms and abbreviations used in this book. If you do not find the term you are looking for, refer to the index or to the *IBM Dictionary of Computing*, SC20-1699.

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).
- *Information Technology Vocabulary*, developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1.SC1).
- *CCITT Sixth Plenary Assembly Orange Book, Terms and Definitions* and working documents published by the International Telecommunication Union, Geneva, 1978.
- Open Software Foundation (OSF).

A

API. Application program interface.

application program interface (API). A functional interface supplied by the operating system or by a separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or the licensed program.

attribute. Information of a particular type concerning an object and appearing in an entry that describes the object in the directory information base (DIB). It denotes the attribute's type and a sequence of one or more attribute values, each accompanied by an integer denoting the value's syntax.

B

binding. A relationship between a client and a server involved in a remote procedure call.

C

CDS. Cell Directory Service.

Cell Directory Service (CDS). A DCE component. A distributed replicated database service that stores names and attributes of resources located in a cell. CDS manages a database of information about the resources in a group of machines called a DCE cell.

certificate. Used to prove your identity. A secure server must have a certificate and a public-private key pair. A certificate is issued and signed by a Certificate Authority (CA).

client. A computer or process that accesses the data, services, or resources of another computer or process on the network. Contrast with *server*.

cipher. A method of transforming text in order to conceal its meaning.

D

data hierarchy. A data structure consisting of sets and subsets such that every subset of a set is of lower rank than the data of the set.

data model. (1) A logical view of the organization of data in a database. (2) In a database, the user's logical view of the data in contrast to the physically stored data, or storage structure. (3) A description of the organization of data in a manner that reflects information structure of an enterprise.

database. A collection of data with a given structure for accepting, storing, and providing, on demand, data for multiple users.

DCE. Distributed Computing Environment.

directory. (1) A logical unit for storing entries under one name (the directory name) in a CDS namespace. Each physical instance of a directory is called a replica. (2) A collection of open systems that cooperates to hold a logical database of information about a set of objects in the real world.

directory schema. The set of rules and constraints concerning directory information tree (DIT) structure, object class definitions, attribute types, and syntaxes that characterize the directory information base (DIB).

directory service. The directory service is a central repository for information about resources in a distributed system.

distinguished name (DN). One of the names of an object, formed from the sequence of RDNs of its object entry and each of its superior entries.

Distributed Computing Environment (DCE). A comprehensive, integrated set of services that supports the development, use, and maintenance of distributed applications. DCE is independent of the operating system and network; it provides interoperability and portability across heterogeneous platforms.

DN. Distinguished name.

E

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

L

LDAP. Lightweight Directory Access Protocol.

Lightweight Directory Access Protocol (LDAP). A client/server protocol for accessing a directory service.

O

object class. An identified family of objects that share certain characteristics. An object class can be specific to one application or shared among a group of applications. An application interprets and uses an entry's class-specific attributes based on the class of the object that the entry describes.

P

private key. Used for the encryption of data. A secure server keeps its private key secret. A secure server sends clients its public key so they can encrypt data to the server. The server then decrypts the data with its private key.

programming interface. The supported method through which customer programs request software services. The programming interface consists of a set of callable services provided with the product.

protocol. A set of semantic and syntactic rules that determines the behavior of functional units in achieving communication.

public key. Used for the encryption of data. A secure server makes its public key widely available so that its clients can encrypt data to send to the server. The server then decrypts the data with its private key.

R

RDN. Relative distinguished name.

referral. An outcome that can be returned by a directory system agent that cannot perform an operation itself. The referral identifies one or more other directory system agents more able to perform the operation.

relative distinguished name (RDN). A component of a DN. It identifies an entry distinctly from any other entries which have the same parent.

S

SASL. Simple Authentication Security Layer.

schema. See *directory schema*.

Secure Sockets Layer (SSL) security. A facility used to protect LDAP access.

server. On a network, the computer that contains programs, data, or provides the facilities that other computers on the network can access. Contrast with *client*.

Simple Authentication Security Layer (SASL). Refers to a method of binding using SSL authentication and the client's certificate identity.

SSL. Secure Sockets Layer.

T

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

X

X.500. The CCITT/ISO standard for the open systems interconnection (OSI) application-layer directory. It allows users to register, store, search, and retrieve information about any objects or resources in a network or distributed system.

X/OPEN Directory Service (XDS). An application program interface that DCE uses to access its directory service components. XDS provides facilities for adding, deleting, and looking up names and their attributes. The XDS library detects the format of the name to be looked up and directs the calls it receives to either GDS or CDS. XDS uses the X/OPEN object management (XOM) API to define and manage its information.

X/OPEN object management (XOM). An interface for creating, deleting, and accessing objects containing information. It is an object-oriented architecture: each

object belongs to a particular class, and classes can be derived from other classes inheriting the characteristics of the original classes. The representation of the object

is transparent to the programmer; the object can be manipulated only through the XOM interface.

XOM. The X/OPEN Object Management API.

Bibliography

This bibliography provides a list of publications that are useful when using the LDAP programming interface. The complete title, order number, and a brief description is given for each publication.

IBM C/C++ Language Publication

- *IBM OS/390 C/C++ Programming Guide*, SC09-2362

This book describes how to develop applications in the C/C++ language in OS/390.

IBM OS/390 Security Server Publication

- *OS/390 Security Server LDAP Server Administration and Usage Guide*, SC24-5861

This book describes how to install, configure, and run the stand-alone LDAP daemon (SLAPD). It is intended for administrators who will maintain the server and database.

IBM OS/390 Cryptographic Services Publication

- *OS/390 Cryptographic Services System Secure Sockets Layer Programming Guide and Reference*, SC24-5877

Contains guidance and reference information that an application programmer needs to use the

System Secure Sockets Layer (SSL) callable programming interfaces. SSL is a communications layer that provides data privacy and integrity, as well as server and client authentication based on public key certificates.

Index

A

- abandoning LDAP operation 27
- accessing RACF information 26
- adding LDAP entry 8, 29
- APIs (Application Program Interfaces)
 - See routines
- Application Program Interfaces (APIs)
 - See routines
- asynchronous LDAP operations 13
- attribute values
 - comparing 9
 - counting 54
 - retrieving 54
- attributes
 - counting 46
 - LDAP 2
 - stepping through 46
 - type 2
- authentication, certificate 1
- authentication, simple 1

B

- bibliography 115
- binding to Directory Service 6
- binding to LDAP server 32
- books, related 115
- breaking down LDAP URL 92

C

- C programming language
 - utility routines 2
- C/C++ programming language
 - LDAP DLL 3
- call-back function 32
- canceling LDAP operation 27
- certificate authentication 1
- changing entry name 77
- changing LDAP entries 72
- changing RDN 9, 77
 - LDAP
 - modifying 77
- character string, deallocating 69
- checking for LDAP URL 92
- classes, LDAP SPI 20
- client API, LDAP 3
- client control 25
- comparing LDAP entries 37
- compiling program 5
- continuation references, retrieving 48

controls

- LDAP 25, 48
 - session 25
- conventions used in book vii
- counting attributes 46
- counting continuation references 48
- counting LDAP entries 48
- counting LDAP handles 54
- counting LDAP values 54
- creating SSL connection 88

D

- data model
 - LDAP 2
- data structure
 - client and server controls 25
- datasets, OS/390 3
- de-initializing LDAP API 7
- deallocating array of LDAP values 54
- deallocating character strings 69
- deallocating LDAP handle 32
- deallocating LDAP URL description 92
- deallocating LDAP values 54
- deallocating memory 80
- deallocating storage 5, 69
- deallocating structures 72
- definitions of terms 111
- deleting LDAP entries 9, 40
- deprecated APIs
 - ldap_bind 32
 - ldap_bind_s 32
 - ldap_modrdn 77
 - ldap_modrdn_s 77
 - ldap_open 57
 - ldap_perror 42
 - ldap_result2error 42
 - listing of 26
- describing error message 42
- directory
 - access protocol, LDAP 1
 - entry
 - naming 2
- Directory Service
 - extracting information from
 - using LDAP 1
 - JNDI interface 20
- distinguished name (DN)
 - getting from LDAP entry 52
 - parsing 52
 - specifying with LDAP 2

DLL (dynamic link library)

See dynamic link library (DLL)

DN

See distinguished name (DN)

documentation, related 115**dynamic link library (DLL)**

C/C++ 3

E**entries 2****entry**

LDAP 2

environment variables 59**error code, returning 42****error handling**

LDAP 4, 5, 11

error message, describing 42**errors**

printing indication of 42

retrieving 42

establishing call-back function 32**extracting information 1****extracting information from results 75****F****first LDAP entry, getting 48****freeing array of LDAP values 54****freeing character strings 69****freeing LDAP handle 32****freeing LDAP URL description 92****freeing LDAP values 54****freeing memory 80****freeing storage 5, 69****freeing structures 72****function, call-back 32****functions, utility**

LDAP 4, 5

G**getting error codes 42****getting first LDAP entry 48****getting LDAP attribute values 54****getting LDAP DNs 52****getting LDAP handles to attribute values 54****getting next attribute type name 46****getting next LDAP entry 48****getting option 57****glossary of terms 111****H****handling errors**

LDAP 4, 5

header files

lber.h 95

ldap.h 96

ldapsl.h 105

I**initialization functions**

LDAP 4

initializing LDAP context 57**initializing SSL 88****interface**

programming, information 110

programming, LDAP 1

interpreting LDAP results 42**interpreting results**

LDAP 5, 10

J**Java Naming and Directory Interface (JNDI)**

using 20

JNDI (Java Naming and Directory Interface)

See Java Naming and Directory Interface (JNDI)

L**lber.h header file 95****LDAP (Lightweight Directory Access Protocol)**

See Lightweight Directory Access Protocol (LDAP)

LDAP service provider interface (SPI)

description 20

LDAP SPI (service provider interface)

See LDAP service provider interface (SPI)

ldap_abandon API 27**ldap_abandon_ext API 27****ldap_add 7****ldap_add API 29****ldap_add_ext API 29****ldap_add_ext_s API 29****ldap_add_s 8****ldap_add_s API 29****ldap_bind 6****ldap_bind API 32****ldap_bind_s 6****ldap_bind_s API 32****ldap_compare 7****ldap_compare API 37****ldap_compare_ext API 37****ldap_compare_ext_s API 37****ldap_compare_s 9****ldap_compare_s API 37****ldap_control_free API 69****ldap_controls_free API 69****ldap_count_attributes API 46**

- ldap_count_entries API 48
- ldap_count_messages API 70
- ldap_count_references API 48
- ldap_count_values API 54
- ldap_count_values_len API 54
- LDAP_DEBUG 13
- ldap_delete 7, 9
- ldap_delete API 40
- ldap_delete_ext API 40
- ldap_delete_ext_s API 40
- ldap_delete_s API 40
- ldap_err2string 12
- ldap_err2string API 42
- ldap_explode_dn API 52
- ldap_first_attribute API 46
- ldap_first_entry API 48
- ldap_first_message API 70
- ldap_first_reference API 48
- ldap_free_urldesc API 92
- ldap_get_dn API 52
- ldap_get_entry_controls_np API 48
- ldap_get_errno 11
- ldap_get_errno API 42
- ldap_get_option API 57
- ldap_get_values API 54
- ldap_get_values_len API 54
- ldap_init 6
- ldap_init API 57
- ldap_is_ldap_url API 92
- ldap_memfree API 69
- ldap_modify 7
- ldap_modify API 72
- ldap_modify_ext API 72
- ldap_modify_ext_s API 72
- ldap_modify_s 8
- ldap_modify_s API 72
- ldap_modrdn API 77
- ldap_modrdn_s 9
- ldap_modrdn_s API 77
- ldap_mods_free API 72
- ldap_msgfree API 80
- ldap_msgid API 80
- ldap_msgtype API 80
- ldap_next_attribute API 46
- ldap_next_entry API 48
- ldap_next_message API 70
- ldap_next_reference API 48
- ldap_open 6
- ldap_open API 57
- ldap_parse_reference_np API 48
- ldap_parse_result API 75
- ldap_parse_sasl_bind_result API 75
- ldap_perror 12
- ldap_perror API 42
- ldap_rename API 77

- ldap_rename_s API 77
- ldap_result 5, 13
- ldap_result API 80
- ldap_result2error 11
- ldap_result2error API 42
- ldap_sasl_bind API 32
- ldap_sasl_bind_s API 32
- ldap_search 5, 7
- ldap_search API 83
- ldap_search_ext API 83
- ldap_search_ext_s API 83
- ldap_search_s 10
- ldap_search_s API 83
- ldap_search_st API 83
- ldap_set_option 13
- ldap_set_option API 57
- ldap_set_option_np API 57
- ldap_set_rebind_proc API 32
- ldap_simple_bind API 32
- ldap_simple_bind_s API 32
- ldap_ssl_client_init API 88
- ldap_ssl_init API 88
- ldap_ssl_start API 88
- ldap_unbind 7
- ldap_unbind API 32
- ldap_unbind_s 7
- ldap_unbind_s API 32
- ldap_url_parse API 92
- ldap_url_search API 92
- ldap_url_search_s API 92
- ldap_url_search_st API 92
- ldap_value_free API 54
- ldap_value_free_len API 54
- ldap.h header file 96
- ldapdelete.c 14
- ldaps.h header file 105
- Lightweight Directory Access Protocol (LDAP)**
 - adding entry 8
 - API functions 3
 - asynchronous operation 13
 - changing RDN of entry 9
 - client for Java 20
 - comparing attribute values 9
 - defining 1
 - deleting entry 9
 - error handling 11
 - example program 14
 - header files
 - lber.h 95
 - ldap.h 96
 - ldaps.h 105
 - interface routines
 - abandoning operation 27
 - adding entry 29
 - binding to server 32
 - checking for URL 92
 - comparing entries 37

Lightweight Directory Access Protocol (LDAP)

(continued)

interface routines (continued)

- controls 25
- counting attributes 46
- counting continuation references 48
- counting entries 48
- counting pointers 54
- counting values 54
- creating SSL connection 88
- deallocating array of values 54
- deallocating character strings 69
- deallocating memory 80
- deallocating storage 69
- deallocating structures 72
- deallocating URL description 92
- deallocating values 54
- deleting entries 40
- deprecated routines 26
- describing error message 42
- establishing call-back function 32
- extracting information from results 75
- freeing array of values 54
- freeing memory 80
- freeing storage 69
- freeing structures 72
- freeing URL description 92
- freeing values 54
- getting option 57
- initializing context 57
- initializing LDAP context 57
- initializing SSL 88
- modifying entries 72
- modifying entry name 77
- modifying RDN 77
- obtaining DNS 52
- obtaining message ID 80
- obtaining message type 80
- opening connection 57
- parsing DNS 52
- parsing URL 92
- printing error 42
- rebinding 32
- retrieving array of server controls 48
- retrieving attribute values 54
- retrieving error codes 42
- retrieving first entry 48
- retrieving list of continuation references 48
- retrieving next attribute type name 46
- retrieving next entry 48
- retrieving pointers to attribute values 54
- returning error code 42
- returning result 80
- searching entries 83
- searching entries with timeout 83
- searching for URL 92
- searching for URL with timeout 92

Lightweight Directory Access Protocol (LDAP)

(continued)

interface routines (continued)

- session controls 25
- setting option 57
- stepping through attributes 46
- stepping through messages 70
- unbinding 32
- waiting for result 80
- ldapdelete.c 14
- listing all sub-entries 10
- modifying entry 8
- program structure 6
- programming 1
- reading attribute values 10
- reading entry's contents 10
- results, getting 10
- synchronous operation 13
- threadsafety 13
- using the API 6
- linking program 5**
- listing sub-entries 10**

M

Makefile

- example 6

message

- stepping through list of 70

model, data

- LDAP 2

modifying entry name 77

modifying LDAP entries 8, 72

modifying RDN 77

multiple operations 13

N

name

- typed 2

nonportable API 57

O

object

- class 2

obtaining LDAP DNS 52

obtaining LDAP message ID 80

obtaining LDAP message type 80

opening LDAP connection 57

option, setting value of LDAP 57

organization of book vii

OS/390 datasets 3

OS/390 shell 3

P

- parsing information from results** 75
- parsing LDAP DNs** 52
- parsing LDAP URL** 92
- primitive operations**
 - LDAP 4, 5
- printing LDAP error** 42
- processing errors** 11
- processing results**
 - LDAP 4, 5
- processing URLs**
 - LDAP 4, 5
- programming interface**
 - LDAP 1
- programming interface information** 110
- protocol**
 - LDAP 1
- publications, related** 115

R

- RACF (Resource Access Control Facility)**
 - See Resource Access Control Facility (RACF)
- RDN (relative distinguished name)**
 - See relative distinguished name (RDN)
- reading attribute values** 10
- reading entry's contents** 10
- rebinding** 32
- referrals** 32
- relative distinguished name (RDN)**
 - changing 9
 - examples of LDAP RDNs 2
 - LDAP
 - modifying 77
 - using with LDAP 2
- removing LDAP entries** 40
- Resource Access Control Facility (RACF)** 26
- results processing**
 - LDAP 4, 5
- results, extracting information from** 75
- results, getting with LDAP** 10
- retrieving array of server controls** 48
- retrieving error codes** 42
- retrieving first LDAP entry** 48
- retrieving LDAP attribute values** 54
- retrieving LDAP entry count** 48
- retrieving next attribute type name** 46
- retrieving next LDAP entry** 48
- retrieving option** 57
- retrieving pointers to attribute values** 54
- returning LDAP error code** 42
- returning LDAP result** 80
- routines**
 - ldap_abandon 27
 - ldap_abandon_ext 27

routines *(continued)*

- ldap_add 29
- ldap_add_ext 29
- ldap_add_ext_s 29
- ldap_add_s 29
- 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 69
- ldap_controls_free 69
- ldap_count_attributes 46
- ldap_count_entries 48
- ldap_count_messages 70
- ldap_count_reference 48
- ldap_count_values 54
- ldap_count_values_len 54
- ldap_delete 40
- ldap_delete_ext 40
- ldap_delete_ext_s 40
- ldap_delete_s 40
- ldap_err2string 42
- ldap_explode_dn 52
- ldap_first_attribute 46
- ldap_first_entry 48
- ldap_first_message 70
- ldap_first_reference 48
- ldap_free_urldesc 92
- ldap_get_dn 52
- ldap_get_entry_controls_np 48
- ldap_get_errno 42
- ldap_get_option 57
- ldap_get_values 54
- ldap_get_values_len 54
- ldap_init 57
- ldap_is_ldap_url 92
- ldap_memfree 69
- ldap_modify 72
- ldap_modify_ext 72
- ldap_modify_ext_s 72
- ldap_modify_s 72
- ldap_modrdn 77
- ldap_modrdn_s 77
- ldap_mods_free 72
- ldap_msgfree 80
- ldap_msgid 80
- ldap_msgtype 80
- ldap_next_attribute 46
- ldap_next_entry 48
- ldap_next_message 70
- ldap_next_reference 48
- ldap_open 57
- ldap_parse_reference_np 48
- ldap_parse_result 75

routines *(continued)*

- ldap_parse_sasl_bind_result 75
- ldap_perror 42
- ldap_rename 77
- ldap_rename_s 77
- ldap_result 80
- ldap_result2error 42
- ldap_search 83
- ldap_search_ext 83
- ldap_search_ext_s 83
- ldap_search_s 83
- ldap_search_st 83
- 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 88
- ldap_ssl_init 88
- ldap_ssl_start 88
- ldap_unbind 32
- ldap_unbind_s 32
- ldap_url_parse 92
- ldap_url_search 92
- ldap_url_search_s 92
- ldap_url_search_st 92
- ldap_value_free 54
- ldap_value_free_len 54

routines, C utility 2

running program 5

S

search

using LDAP 5

search results, counting 48

searching for LDAP URL 92

searching LDAP entries 83

searching LDAP entries with timeout 83

Secure Sockets Layer (SSL)

creating connection 88

initializing 88

security

supported by LDAP 1

server control 25

server, LDAP

binding to 32

unbinding from 32

service provider interface (SPI), LDAP

See LDAP service provider interface (SPI)

session settings 59

setting LDAP option 57

shell, UNIX 3

simple authentication 1

SPI (service provider interface), LDAP

See LDAP service provider interface (SPI)

SSL (Secure Sockets Layer)

See Secure Sockets Layer (SSL)

standard error stream 13, 42

starting SSL 88

stepping through LDAP attributes 46

storage

deallocating 69

freeing 5

structure, LDAP program 6

sub-entries, listing 10

synchronous LDAP operations 13

T

TCP/IP

See Transmission Control Protocol/Internet Protocol (TCP/IP)

termination functions

LDAP 4

terms, glossary of 111

threadsafety

LDAP API 13

tracing

disabling 5

enabling 5, 13

Transmission Control Protocol/Internet Protocol (TCP/IP)

LDAP use of 21

typed names 2

U

unbinding from LDAP server 32

unbinding LDAP API 7

Universal Resource Locator (URL)

breaking down LDAP 92

checking for 92

deallocating LDAP 92

searching for LDAP 92

URL

See Universal Resource Locator (URL)

URL processing

LDAP 4, 5

utility functions

LDAP 4, 5

utility routines, C 2

W

waiting for LDAP result 80

X

X.500, naming concepts 2

Readers' Comments

OS/390®
Security Server
LDAP Client Application Development
Guide and Reference
Publication No. SC24-5878-01

You may use this form to report errors, to suggest improvements, or to express your opinion on the appearance, organization, or completeness of this book.

Date: _____

IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note

Report system problems to your IBM representative or the IBM branch office serving you.
U.S. customers can order publications by calling the IBM Software Manufacturing Solutions at
1-800-879-2755.

In addition to using this postage-paid form, you may send your comments by:

FAX	1-607-752-2327	Internet	pubrcf@vnet.ibm.com
IBM Mail	USIB2L8Z@IBMMAIL	IBMLink	GDLVME(PUBRCF)

Would you like a reply? ☐ **YES** ☐ **NO** If yes, please tell us the type of response you prefer.

☐ Electronic address: _____

☐ FAX number: _____

☐ Mail: (Please fill in your name and address below.)

Name

Address

Company or Organization

Phone No.



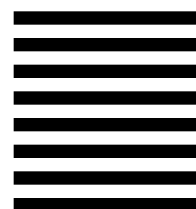
Fold and Tape

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

Department G60
International Business Machines Corporation
Information Development
1701 North Street
ENDICOTT NY 13760-5553



Fold and Tape

Please do not staple

Fold and Tape



Program Number: 5647-A01



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC24-5878-01





OS/390 Security Server

LDAP Client Application Development Guide and Reference