z/OS

IBM Tivoli Directory Server Client Programming for z/OS

Version 2 Release 2

SA23-2295-01

Note
Before using this information and the product it supports, read the information in "Notices" on page 311.

This edition applies to Version 2 Release 2 of z/OS (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

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.

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

This product includes software developed by NEC Systems Laboratory.

Contents

Tables

About this document

This document describes the Lightweight Directory Access Protocol (LDAP) client application, which is part of IBM® Tivoli® Directory Server for z/OS®, and supports z/OS (5650-ZOS).

Intended audience

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

Conventions used in this document

This document uses the following typographic conventions:

Bold **Bold** words or characters represent API names, functions, routines, utility names, and system elements that you must enter into the system literally, such as commands and options.

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

`Example font`
Path names, attributes, environment variables, parameter values, 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.

... 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 non-blank character on the line to be continued, and continue the command on the next line.

Where to find more information

When possible, this information uses cross-document links that go directly to the topic in reference using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see *z/OS V2R2 Information Roadmap*.

To find the complete z/OS library, including the z/OS Information Center, see z/OS Internet Library (http://www.ibm.com/systems/z/os/zos/bkserv/).

Internet sources

The following resources are available through the internet to provide additional information about the z/OS library and other security-related topics:

- **Online library**

  To view and print online versions of the z/OS publications, use this address:

  `http://www.ibm.com/systems/z/os/zos/bkserv/`

- **Redbooks®**

  The documents known as IBM Redbooks that are produced by the International Technical Support Organization (ITSO) are available at the following address:

  `http://www.redbooks.ibm.com`

How to send your comments to IBM

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or provide any other feedback that you have.

Use one of the following methods to send your comments:

1. Send an email to mhvrcfs@us.ibm.com.
2. Send an email from the "Contact us" web page for z/OS (http://www.ibm.com/systems/z/os/zos/webqs.html).

Include the following information:

- Your name and address.
- Your email address.
- Your telephone or fax number.
- The publication title and order number:
  z/OS V2R2 IBM Tivoli Directory Server Client Programming for z/OS SA23-2295-01
- The topic and page number that is related to your comment.
- The text of your comment.

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

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

Do not use the feedback methods that are listed for sending comments. Instead, take one of the following actions:

- Contact your IBM service representative.
- Call IBM technical support.
- Visit the IBM Support Portal at z/OS Support Portal (http://www-947.ibm.com/systems/support/z/zos/).

z/OS Version 2 Release 2 summary of changes for IBM Tivoli Directory Server Client Programming for z/OS

The following changes are made to z/OS Version 2 Release 2 (V2R2).

Changed

- The **-L** option is updated for the "ldapsearch utility" on page 296. See Table 13 on page 296.
- LDAP routine, "ldap_ssl_client_init()" on page 198, is updated to indicate that LDAP no longer supports SSL V2 protocol and that SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 protocols are supported.
- Deprecated LDAP routine, "ldap_ssl_start()" on page 231, is updated to indicate that LDAP no longer supports SSL V2 protocol and that SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 protocols are supported.
- "Using environment variables to control SSL/TLS settings" on page 262 is updated to indicate that LDAP no longer supports SSL V2 protocol and that SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 protocols are supported.

Summary of changes for z/OS Version 2 Release 1

See the following publications for all enhancements to z/OS Version 2 Release 1 (V2R1):

- *z/OS V2R2 Migration*
- *z/OS Planning for Installation*
- *z/OS Summary of Message and Interface Changes*
- *z/OS V2R2 Introduction and Release Guide*

Chapter 1. LDAP programming

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

The LDAP support in z/OS is for client access to Directory Services that accept the LDAP protocol. The LDAP client allows programs running on z/OS UNIX System Services to store and extract information into and from a Directory Service. The LDAP server can be used to store and extract information about z/OS using the LDAP protocol. For more information, see *z/OS IBM Tivoli Directory Server Administration and Use for z/OS*.

How LDAP is defined

The LDAP protocol is defined by a number of Internet Engineering Task Force (IETF) request for comments (RFCs).

Current RFCs

The z/OS LDAP client supports all or parts of the following Internet Engineering Task Force (IETF) request for comments (RFCs):

- RFC 1738: *Uniform Resource Locators (URL)*
- RFC 1823: *The LDAP Application Program Interface*
- RFC 1928: *SOCKS Protocol Version 5*
- RFC 1929: *Username/Password Authentication for SOCKS V5*
- RFC 2052: *A DNS RR for specifying the location of services (DNS SRV)*
- RFC 2195: *IMAP/POP AUTHorize Extension for Simple Challenge/Response*
- RFC 2222: *Simple Authentication and Security Layer (SASL)*
- RFC 2251: *Lightweight Directory Access Protocol (v3)*
- RFC 2252: *Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions*
- RFC 2253: *Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names*
- RFC 2254: *The String Representation of LDAP Search Filters*
- RFC 2255: *The LDAP URL Format*
- RFC 2256: *A Summary of the X.500 (96) User Schema for use with LDAPv3*
- RFC 2373: *IP Version 6 Addressing Architecture*
- RFC 2696: *LDAP Control Extension for Simple Paged Results Manipulation*
- RFC 2732: *Format for Literal IPv6 Addresses in URLs*
- RFC 2829: *Authentication Methods for LDAP*
- RFC 2830: *Lightweight Directory Access Protocol (v3): Extension for Transport Layer Security*
- RFC 2831: *Using Digest Authentication as a SASL Mechanism*
- RFC 2849: *The LDAP Data Interchange Format (LDIF)*
- RFC 2891: *LDAP Control Extension for Server Side Sorting of Search Results*

Draft RFCs

The z/OS LDAP client supports all or parts of the following request for comment (RFC) draft:

- *Password Policy for LDAP directories*

Superseded RFCs

The following obsolete RFCs were implemented by the z/OS LDAP client and server but have been superseded by current RFCs:

- RFC 1777: *Lightweight Directory Access Protocol*
- RFC 1778: *The String Representation of Standard Attribute Syntaxes*
- RFC 1779: *A String Representation of Distinguished Names*
- RFC 1959: *An LDAP URL Format*
- RFC 1960: *A String Representation of LDAP Search Filters*

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* that are required to be present in the entry including 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* that describes the type of the attribute values. Examples of attribute syntaxes include **Directory String** and **Octet String**.

To summarize, the directory is made up of entries. Each entry contains a set of attributes. These attributes can be single or multi-valued (have one or more values that are associated with them). The object class of an entry determines the set of attributes that must and the set of attributes that might exist in the entry.

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

LDAP names

The LDAP protocol and APIs use *typed* names to identify directory entries. In contrast, the Domain Name Service (DNS) uses *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 LDAP format for this DN is:

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

An LDAP DN is specified as a null-terminated character string in a right-to-left fashion (right-to-left refers to the ordering of RDNs from highest to lowest in the directory hierarchy). Note that embedded spaces are taken as part of the attribute value for RDNs and do not require quotation marks. Also, note that RDNs are separated by commas (,) and attribute type/value pairs within an RDN are separated by plus (+) signs. (See RFC 2253: *UTF-8 String Representation of Distinguished Names* for more information.)

Function overview

The LDAP client API is provided in a C DLL that is loaded at run time by applications using the LDAP API. The DLL that externalizes the LDAP programming interfaces is called **GLDCLDAP** for 31-bit applications and **GLDCLD64** for 64-bit applications. The DLL can be loaded into LPA, specified in the link list, or included in the **STEPLIB** for the job. The LDAP API consists of C language functions.

All function names begin with the prefix **ldap_**. Synchronous versions of the APIs have a suffix of **_s**, for example, **ldap_add_s()**. The **_np** suffix indicates that the API is non-portable. That is, the API is not defined in an RFC and might not be available with other LDAP implementations. The **_ext** suffix indicates that the API is an enhanced version of an existing API. For example, **ldap_search_ext()** is an enhanced version of **ldap_search()**.

For detailed information about each LDAP API, see Chapter 2, "LDAP routines," on page 23 and Chapter 3, "Deprecated LDAP routines," on page 221.

ASCII support

EBCDIC is the default for the LDAP client run time. In EBCDIC mode, all text data is in the local EBCDIC code page. Text data for requests that are sent to the LDAP server is converted from EBCDIC to UTF-8 and text data that is received from the LDAP server is converted from UTF-8 to EBCDIC. The EBCDIC code page is based on the value that is specified for the **setlocale()** API routine. The IBM-1047 code page is used if the application does not call **setlocale()** to set the current locale.

UTF-8 I/O mode is set by calling the **ldap_set_option()** or **ldap_set_option_np()** routine and turning on the `LDAP_OPT_UTF8_IO` option. In this mode, text data for LDAP client operations is in UTF-8. This data includes host names, user names, passwords, and error messages. Text data for requests that are sent to the LDAP server is assumed to be in UTF-8 and is not converted. Similarly, text data that is received from the LDAP server is returned to the application in UTF-8.

Native ASCII mode is set by defining the `LDAP_LIBASCII` compiler variable before including the `ldap.h` header file. In this mode, all text data is in UTF-8. This includes text data for LDAP client operations, and data sent to the LDAP server or received from the LDAP server. When the `LDAP_LIBASCII` compiler variable is defined, the `LDAP_OPT_UTF8_IO` option is automatically set whenever an LDAP

handle is created. Note the interfaces between the LDAP client run time and the underlying operating system routines use EBCDIC. This means that the UTF-8 text data for LDAP client operations must not contain any characters that cannot be represented in the local EBCDIC code page.

Compiling, linking, and running a program

The LDAP API is supplied in a C DLL that is loaded at program run time, enabling a program to call the functions of the interface. The following rules apply to compiling and link-editing programs that use the LDAP API.

Rules

1. Include the `ldap.h` header file in all C or C++ source files that make calls to the LDAP API. If you use SSL/TLS, you must include the `ldapssl.h` include file after the `ldap.h` include file.
2. When compiling, specify `-Wc,DLL` on the compile of all modules that make calls to the LDAP API.
3. When linking the program, specify `-Wl,DLL` and include an LDAP sidefile as one of the files to be linked with the program. The LDAP sidefiles are:
   - `/usr/lib/GLDCLDAP.x` for 31-bit applications
   - `/usr/lib/GLDCLD64.x` for 64-bit applications
4. Ensure that your application has `POSIX(ON)` so it can use the LDAP API.
5. When running the program, ensure that the LDAP DLL is accessible. The DLL is supplied in the `SYS1.SIEALNKE` data set.
   - The 31-bit DLL is `GLDCLDAP`.
   - The 64-bit DLL is `GLDCLD64`.
6. Call the **setlocale()** routine to set the current locale before the first call to an LDAP API.
7. If you are using SSL/TLS or Kerberos authentication, you must use the `SYS1.SIEALNKE` data set.

Makefile and README files are shipped in `/usr/lpp/ldap/examples` to explain how to build the LDAP sample applications. You might be able to use this information as a base for building your LDAP application.

Using TSO and batch jobs

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

- Data set *GLDHLQ*.SGLDHDRC contains the LDAP header files.
- Data set *GLDHLQ*.SGLDEXPC contains the sidefiles.
- `POSIX(ON)` must be specified as a runtime option because the default for this environment is `POSIX(OFF)`.

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. Before initialization, SIGPIPE signals should be set to be ignored or a signal handler should be defined. TCP/IP functions can cause SIGPIPE signals. When the signal is ignored, TCP/IP reflects the signal as an EPIPE error for the TCP/IP functions.
2. Initialize the LDAP programming interface and the connection to the directory server that accepts the LDAP protocol using **ldap_init()**.
3. Bind to the Directory Service to establish an identity with the directory server by using **ldap_simple_bind(), ldap_simple_bind_s(), ldap_sasl_bind()**, or **ldap_sasl_bind_s()**.
4. Perform LDAP operations such as add, modify, delete, compare, and search.
5. When all LDAP operations are completed, unbind the LDAP programming interface using **ldap_unbind()** or **ldap_unbind_s()**.

   Note:

   a. **ldap_unbind_s()** is identical in function to **ldap_unbind()** and is provided as a convenience for those programs that do only synchronous operations so that the unbind does not appear to be an asynchronous operation. All unbind operations are synchronous.

   b. After the **ldap_unbind()** or **ldap_unbind_s()** function returns, the LDAP handle that was returned by **ldap_init()** is no longer valid and must not be used.

   c. To terminate the connection with an LDAP server, it is necessary to unbind, regardless of whether an explicit bind was done.

It is acceptable to perform more than one **ldap_init()** within the same program. More than one LDAP handle can be allocated at the same time. This, however, causes 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.

Authentication methods

Five authentication methods are supported for checking client access to LDAP directory services. They are:

- Simple authentication
- Certificate authentication
- Kerberos credentials authentication
- CRAM-MD5 authentication
- DIGEST-MD5 authentication

For each supported authentication method, Secure Socket Layer (SSL) or Transport Layer Security (TLS) can be used to secure the socket connection between the client and the server by encrypting the data that is transferred over the connection. TLS is based on SSL V3. Through a protocol handshake between the client and server, the choice of TLS or SSL is decided. (TLS is the preferred protocol.)

The supported authentication methods are available through the **ldap_sasl_bind()** routine. (For details, see "ldap_sasl_bind(), ldap_sasl_bind_s()" on page 157.) Each supported authentication method is described briefly as follows:

Simple authentication
A user ID and password are sent (in clear text) from the client to the server to establish who is contacting the LDAP server for information. Mutual authentication is not performed. The server verifies the identity of the client but the client has no way to verify the identity of the server. Simple authentication is also referred to as *simple bind*. The **ldap_simple_bind()** and **ldap_simple_bind_s()** routines can be used to perform a *simple bind*. The **ldap_sasl_bind()** and **ldap_sasl_bind_s()** routines can be used for simple binds as well by passing in `NULL` on the *mechanism* parameter.

Certificate authentication
The identity from the client certificate sent to the LDAP server on an SSL/TLS socket connection is used to establish who is contacting the LDAP server for information. SSL or TLS must be configured on the LDAP server. Certificate authentication is also referred to as *SASL EXTERNAL bind* and is provided by the **ldap_sasl_bind()** and **ldap_sasl_bind_s()** routines.

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

CRAM-MD5 authentication and DIGEST-MD5 authentication
CRAM-MD5 authentication and DIGEST-MD5 authentication are each accomplished in a series of challenges and responses between the client application and server. The response from the client application to the server has a hashed password that is calculated by using an algorithm that is known by both the client application and server. The server checks to make certain that the authentication is correct by calculating its own password hash and comparing it to the client-calculated password hash. Both CRAM-MD5 and DIGEST-MD5 authentications are provided by the **ldap_sasl_bind()** and **ldap_sasl_bind_s()** routines.

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 set up while the results of **ldap_search()** are the most complex to interpret. It is not surprising that these deal with adding or changing and retrieving directory entry contents. An example of a call to each LDAP operation is shown here along with a short explanation, where needed. See Chapter 2, “LDAP routines,” on page 23 for details on the parameters to each LDAP function in the LDAP API.

Adding an entry

Example:

```
modifications = (LDAPMod **)calloc(5, sizeof(LDAPMod *));

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

modifications[0]->mod_type = "objectClass";
modifications[0]->mod_values = (char **)calloc(2, sizeof(char *));
modifications[0]->mod_values[0] = "person";
```

```
modifications[1]->mod_type = "cn";
modifications[1]->mod_values = (char **)calloc(2, sizeof(char *));
modifications[1]->mod_values[0] = "John Doe";
modifications[2]->mod_type = "sn";
modifications[2]->mod_values = (char **)calloc(2, sizeof(char *));
modifications[2]->mod_values[0] = "Doe";
modifications[3]->mod_type = "description";
modifications[3]->mod_values = (char **)calloc(2, sizeof(char *));
modifications[3]->mod_values[0] = "This is John Doe";
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.

To supply a binary attribute, use the *pointer-length* form of input. Set the `mod_op` field of the attribute to `LDAP_MOD_ADD | LDAP_MOD_BVALUES` to indicate that the passed value is binary and in *pointer-length* form. The data is sent to the LDAP server without modification.

When the `LDAP_OPT_UTF8_IO` option is set to `LDAP_OPT_OFF`, the value is supplied as a null-terminated character string in the code set of the current locale. The data is converted to wire protocol before being sent to the LDAP server.

When the `LDAP_OPT_UTF8_IO` option is set to `LDAP_OPT_ON`, the value is supplied as a null-terminated UTF-8 character string. The data is *not* converted to wire protocol before being sent to the LDAP server.

Modifying an entry

Example:

```
modifications = (LDAPMod **)calloc(4, sizeof(LDAPMod *));

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

modifications[0]->mod_op = LDAP_MOD_DELETE;
modifications[0]->mod_type = "description";
modifications[0]->mod_values = (char **)calloc(1, sizeof(char *));
modifications[1]->mod_op = LDAP_MOD_ADD;
modifications[1]->mod_type = "telephoneNumber";
modifications[1]->mod_values = (char **)calloc(2, sizeof(char *));
modifications[1]->mod_values[0] = "1-607-123-4567";
modifications[2]->mod_op = LDAP_MOD_REPLACE;
modifications[2]->mod_type = "sn";
modifications[2]->mod_values = (char **)calloc(2, sizeof(char *));
modifications[2]->mod_values[0] = "Doe, Jr";
rc = ldap_modify_s(ld,
        "cn=John Doe, ou=Marketing, o=Acme International, c=US",
        modifications);
```

The same modifications array construct that was used for an add operation is used for performing a modify operation. The difference is that the `mod_op` field can take on values of `LDAP_MOD_ADD`, `LDAP_MOD_REPLACE`, or `LDAP_MOD_DELETE`. Like **ldap_add()**, you can perform a `bitwise` **`OR`** operation to assign `LDAP_MOD_BVALUES` to 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()**.

Deleting an entire entry

Example:

```
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 fails if the entry to be deleted contains any subentries 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 to wait for the results of the operation.

Changing the RDN of an entry and relocating the entry

Example:

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

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

- `"cn=John Doe, ou=Marketing, o=Acme International, c=US"` is the DN of the entry to be renamed.
- `"cn=Jonathan Doe"` is the new value of the RDN for the renamed entry.
- `"ou=Sales, o=Acme International, c=US"` is the DN of the new superior (parent) node under which the entry is moved; if no relocation is being performed, this parameter should be `NULL`.
- `1, NULL, NULL` indicates that the old RDN value should be deleted from the renamed entry and that the client and server controls that are set in the handle should be used.

When no controls are present, each respective parameter should be set to `NULL`. The X.500 data model states that the attribute types and values that comprise the RDN of an entry are also part of the attribute types and values of the entry itself. When the RDN of an entry is modified, it is the option of the program to specify whether the attribute values that made up the old RDN be retained as attribute types and values of the renamed entry.

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

Example:

```
rc = ldap_compare_s(ld,
        "cn=Jonathan Doe, ou=Marketing, o=Acme International, c=US",
        "telephoneNumber",
        "1-607-555-1234");
```

This operation compares the supplied value (`"1-607-555-1234"`) to all the values of the `telephoneNumber` 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 `telephoneNumber` attribute's values match, `LDAP_COMPARE_FALSE` is returned. If the attribute does not exist or some other error occurs, an appropriate error code is returned.

Reading a directory entry's contents

Example:

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

Listing the objectClass attribute values for all entries directly below a given entry

Example:

```
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);
```

Reading the objectClass attribute values for all entries below a given entry

Example:

```
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, 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 subentries. When `NULL` is specified for the attributes parameter, all attribute types and values are returned in the results sent to the client program.

Getting results

The LDAP results processing functions can be used to interpret the results that are 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 typically 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.

Example: An example of this type of processing is:

```
rc = ldap_search_s(ld,
        "ou=Marketing, o=Acme International, c=US",
        LDAP_SCOPE_SUBTREE,
        "(|(cn=Jane*)(cn=Jon*))",
        NULL, 0, &res);

for (entry = ldap_first_entry(ld, res);
                    entry != NULL;
                    entry = ldap_next_entry( ld, entry)) {
```

```
    dn = ldap_get_dn( ld, entry );
    printf( "Entry: %s\n", dn );
    ldap_memfree( dn );

    for (attrtype = ldap_first_attribute( ld, entry, &ber);
                  attrtype != NULL;
                  attrtype = ldap_next_attribute(ld, entry, ber)) {
        values = ldap_get_values(ld, entry, attrtype);
        if (values != NULL) {
            for (i = 0; values[i] != NULL; i++)
                printf(" %s = %s\n", attrtype, values[i]);

            ldap_value_free(values);
        }
        ldap_memfree(attrtype);
    }
}

ldap_msgfree(res);
```

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 converts the returned results into a null-terminated string in the code set of the current locale unless the LDAP_OPT_UTF8_IO option is set for the LDAP handle. If the data is binary data or if conversions should be avoided, the **ldap_get_values_len()** routine must be used. The data is then supplied in *pointer-length* format and no conversions are performed.

Referrals

When a client requests information from a server that does not hold the needed data, the server can pass back one or more referrals that indicate other servers to contact. The client can then request the information from the referenced servers. The LDAP client follows referrals if the LDAP_OPT_REFERRALS option is set for the LDAP handle. (This is the default.) Otherwise, the referrals are returned to the application for processing. The LDAP_OPT_REFHOPLIMIT option sets a limit on the number of nested referrals that are followed.

The LDAP client supports referral values that are LDAP URLs (Uniform Resource Locators) and ignores any other referral values. The format of an LDAP URL is described in "ldap_url_parse()" on page 207. The *host* part of the URL is required to identify the server to which to send the referral. A secure LDAP URL (one specifying **ldaps** for the scheme) is used only if the LDAP handle is using an SSL connection. A non-secure LDAP URL (one specifying **ldap** for the scheme) is used for SSL and non-SSL connections. The default port (389 for non-secure connections or 636 for secure connections) is used if the LDAP URL does not specify a port. Because a non-secure LDAP URL can be used with both non-SSL and SSL connections, an explicit port specification in the URL does not work for both connection types because the LDAP server requires different ports for non-SSL and SSL connections. If you are using non-default ports for the LDAP server, the referral definition should contain values for both **ldap:** and **ldaps:** schemes.

Upon completion of referral processing, any unfollowed referrals are appended to the error string in the result message. The result code is set to LDAP_SUCCESS if all the referrals were processed successfully. Otherwise, the result code is set to the error code for the first referral failure.

Servers present the referrals differently depending on the LDAP protocol version being used by the client. Referrals for the LDAP Version 2 protocol are returned in the error string, as the protocol does not provide a specific mechanism for indicating referrals. Referrals for the LDAP Version 3 protocol are returned in search reference messages and in the result message with a result code of LDAP_REFERRAL.

Using LDAP Version 2 referrals

LDAP Version 2 referrals are returned as part of the error string in the result message. Because clients do not examine the error string for results indicating LDAP_SUCCESS, the server returns a result code of LDAP_PARTIAL_RESULTS instead of LDAP_SUCCESS to indicate the presence of referral information in the error string. Referral information can be returned in the error string for any result code other than LDAP_SUCCESS.

The referral information is at the end of the error string and looks like the following:

```
Referral:\n
ldap://hostname1:port1/dn\n
ldaps://hostname2:port2/dn\n
...
```

where \n indicates a new-line character.

Multiple referrals are present only for partial search results when it is necessary to contact more than one additional server to complete the entire request. This indicates that multiple referral definitions were found that matched the search criteria. The client contacts every server that is presented in the list to continue the search request. Only the first referral value is returned for each referral definition because there is no way to distinguish between a single referral definition with multiple referral values and multiple referral definitions.

Using LDAP Version 3 referrals

The LDAP Version 3 protocol defines referrals as part of the protocol. There are two methods of passing back referral information: referrals and search continuation references.

- **Referrals:** The LDAP_REFERRAL result code is returned by the server to indicate that the server does not hold the target entry of the request. The referral field is present in the result message and indicates another server (or set of servers) to contact. Referrals can be returned in response to any operation except abandon and unbind. When multiple referrals are present in a given referral response, each one must be equally capable of being used to continue the operation.
- **Search continuation references:** A referral is not returned in the result for a one-level or subtree search in which the search scope spans multiple referral objects. Instead, one or more search continuation references are returned. Search continuation references are intermixed with search entries. Each search continuation reference contains a referral to another server (or set of servers) to contact and represents a subtree of the namespace which potentially satisfies the search criteria. When multiple referrals are present in a given search continuation reference, each one must be equally capable of being used to continue the operation.

The LDAP Version 3 protocol provides the manageDsaIT control to allow the client to operate on the referral object instead of the real object. When this control is

included in the client request, the server does not present any referrals or search continuation references, but instead treats the referral objects as normal objects.

Rebinding while following referrals

When the LDAP client follows a referral to a different LDAP server, it must bind to that server. To this, the client must have the proper credentials available to pass to the target LDAP server. Normally, these credentials are passed on the **ldap_simple_bind()** or **ldap_sasl_bind()** function invocation. During referral 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 rebind routine set by **ldap_set_rebind_proc()** or the `LDAP_OPT_REBIND_FN` option is defined as follows:

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

The rebind routine set by the `LDAP_OPT_EXT_REBIND_FN` option is defined as follows:

```
int ext_rebind_proc (
    LDAP *          ld,
    int             msgtype,
    const char *    host,
    const char *    object,
    int             freeit,
    int *           authmethodp,
    char **         dnp,
    char **         passwdp,
    char **         mechanismp,
    BerVal **       credentialsp,
    LDAPControl *** serverctrlsp,
    LDAPControl *** clientctrlsp)
```

When the rebind routine is invoked and the `freeit` parameter is `0`, the rebind routine should set the return values before returning to the caller. The only supported authentication methods for rebinding are `LDAP_AUTH_SIMPLE` and `LDAP_AUTH_SASL`. An anonymous bind is done if an unsupported authentication method is specified.

The `ld` parameter provides the LDAP handle for the request resulting in the referral. This handle can be used to send an unauthenticated request to the target LDAP server (for example, a search request to retrieve attributes from the root DSE). An error is returned if an attempt is made to bind to the server, to abandon active requests, or to unbind the handle.

The `msgtype` parameter provides the message type for the request resulting in the referral. The `host` and `object` parameters provide the host name and the distinguished name for the referral. The text strings are in UTF-8 or the local EBCDIC code page as determined by the `LDAP_OPT_UTF8_IO` option for the LDAP handle. The `object` parameter is `NULL` if there is no distinguished name available.

For the LDAP_AUTH_SIMPLE authentication method, the dnp parameter should be set to the distinguished name for the bind and the passwdp parameter should be set to the password for the bind. The SASL authentication return values are ignored. The text strings must be in UTF-8 or the local EBCDIC code page as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle.

For the LDAP_AUTH_SASL authentication method, the dnp, mechanismp, credentialsp, serverctrlsp and clientctrlsp parameters should be set as described for the **ldap_sasl_bind()** routine. The passwdp return value is ignored. The text strings must be in UTF-8 or the local EBCDIC code page as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle.

When the LDAP_AUTH_SASL authentication method is specified by the **rebind_proc()** routine, the GSSAPI SASL mechanism is used with the default Kerberos credentials. The extended rebind routine should be used if another LDAP_AUTH_SASL authentication method is needed.

When the LDAP_AUTH_SASL authentication method is specified by the **ext_rebind_proc()** routine, any of the supported SASL mechanisms can be used.

When the rebind routine is invoked and the freeit parameter is nonzero, the rebind routine should release any storage that was acquired by the previous call to the rebind procedure. The dnp, passwdp, authmethodp, mechanismp, credentialsp, serverctrlsp and clientctrlsp parameters are the values returned by the previous call to the rebind procedure.

The rebind routine should return LDAP_SUCCESS if the return fields were successfully set. Otherwise, the rebind routine should return one of the error codes in the ldap.h include file. An error return causes the current LDAP operation to be stopped and the error is returned to the original caller. The function return value is ignored when the rebind routine is called to release storage.

When processing a bind referral and no rebind procedure is defined, the LDAP client run time uses the credentials from the original bind request to bind to the target server. For any other type of request, the LDAP client run time performs an unauthenticated bind when no rebind procedure is defined.

Error processing

The following are routines that are used in the LDAP programming interface for handling errors that are returned from LDAP operations:

- **ldap_err2string()**
- **ldap_get_errno()**
- **ldap_get_lderrno()**
- **ldap_get_option()**
- **ldap_parse_pwdpolicy_response()**
- **ldap_parse_result()**
- **ldap_pwdpolicy_err2string()**

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_lderrno()

The **ldap_get_lderrno()** routine returns the most recent LDAP error code, error message, and matched distinguished name (DN) that was logged by the LDAP programming interface against an LDAP handle.

Be careful when using **ldap_get_lderrno()** in a multi-threaded environment where the LDAP handle is shared by multiple threads. If an LDAP operation completes on another thread before **ldap_get_lderrno()** examines the error code, error message, or matched distinguished name (DN), the values that are returned by **ldap_get_lderrno()** reflect the result of the LDAP operation on the other thread. Use the **ldap_parse_result()** and **ldap_err2string()** routines in these cases.

Using ldap_get_errno() and ldap_parse_result()

The most basic error handling routine in the LDAP API is **ldap_get_errno()**. This routine returns the most recent error code that was logged by the LDAP programming interface against an LDAP handle. In the case of LDAP operations that result in errors, the error code value that was returned from the directory server can be obtained by calling **ldap_parse_result()**, passing in the `LDAPMessage` that was returned from the LDAP operation. There is a subtle difference between using **ldap_get_errno()** and **ldap_parse_result()** for asynchronous operations. For asynchronous operations, if an error occurs during the process of sending the request to the directory server, you must use **ldap_get_errno()** to obtain the error value. Use the **ldap_parse_result()** call after a **ldap_result()** call is complete. In the case of synchronous operations, either routine can be used. In addition, the synchronous routines also return the error code value for the programmer's convenience.

Be careful when using **ldap_get_errno()** in a multi-threaded environment where the LDAP handle is shared by multiple threads. If an LDAP operation completes on another thread before **ldap_get_errno()** examines the error code on the current thread, the error code that is returned by **ldap_get_errno()** reflects the result of the LDAP operation on the other thread. Use the **ldap_parse_result()** and **ldap_err2string()** calls in these cases.

Using ldap_err2string() and ldap_get_option()

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

The **ldap_get_option()** routine, when specified with the `LDAP_OPT_ERROR_NUMBER` and `LDAP_OPT_ERROR_STRING` values, obtains the LDAP error code and error message. These can then be issued in a message containing the text returned by **ldap_err2string()** on the standard error stream.

Be careful when using **ldap_get_option()** in a multi-threaded environment where the LDAP handle is shared by multiple threads. If an LDAP operation completes on another thread before **ldap_get_option()** examines the error code or error message values on the current thread, the values returned by **ldap_get_option()** reflect the result of the LDAP operation on the other thread. Use the **ldap_parse_result()** and **ldap_err2string()** calls in these cases.

Using ldap_parse_pwdpolicy_response() and ldap_pwdpolicy_err2string()

The **ldap_parse_pwdpolicy_response()** routine parses the password policy control response returned from the LDAP server and returns the password policy error code, warning code, and warning result value.

The **ldap_pwdpolicy_err2string()** routine, given an error or warning code from the password policy control response, returns a null-terminated character string that provides a textual description of the password policy error or warning.

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 that can be used to define *client* controls. The client controls affect the behavior of the LDAP client library, and are never sent to the server.

A common data structure is used to represent both server controls and client controls:

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

The `LDAPControl` fields have the following definitions:

ldctl_oid
: Specifies the control type as a null-terminated character string in either the local EBCDIC code page or UTF-8 as determined by the `LDAP_OPT_UTF8_IO` option in the LDAP handle for the request that references the control. The control type is a numeric OID with no leading, trailing, or embedded white space characters.

ldctl_value
: Specifies the data associated with the control (if any). An error is returned if the data length is greater than `2147483647`. To specify a zero-length value, set `ldctl_value.bv_len` to `0` 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`.

  The data format depends on the control type. A text string for a server control is in UTF-8. A text string for a client control is in UTF-8 or the local code page as determined by the `LDAP_OPT_UTF8_IO` option for the LDAP handle associated with the request referencing the control. A binary value for a server or client control is formatted as determined by the control OID.

ldctl_iscritical
: Specifies whether the control is critical. If this field is nonzero (critical), the operation is performed only if the control is appropriate for the operation and it is recognized and supported by the server for server controls or the client for client controls. Otherwise, the operation is not performed.

If this field is 0 (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 for server controls or the client for client controls. Otherwise, the control is 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 that perform LDAP operations accept a list of controls (for example, **ldap_search_ext()**). Alternatively, a list of controls that affect 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 that perform LDAP operations. So although **ldap_search()**, for example, does not accept control list parameters, it includes a server control on its request if there was a server control setup through **ldap_set_option()**.

Supported client controls

Currently, the only client controls supported by this library are:

- ibm-serverHandledSearchRequest
- ibm-saslBindDigestUserName
- ibm-saslBindCramUserName
- ibm-saslBindDigestRealmName

Note the object identifier for ibm-saslBindCramUserName is the same as the object identifier for ibm-saslBindDigestUserName.

ibm-serverHandledSearchRequest

Name: ibm-serverHandledSearchRequest

Numeric OID:
1.3.18.0.2.10.7

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

Criticality:
TRUE or FALSE. If TRUE, operations that do not support this control fail with LDAP_UNAVAILABLE_CRITICAL_EXTENSION. If FALSE, operations that do not support this control ignore its presence and process the request.

Value: An ASN.1-encoded sequence as follows:

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

Example:

The following is an example of defining an ibm-serverHandledSearchRequest control.

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

Meaning:

If the control is not present, the search request can be handled from the cache. If the search request is not cached, the search is passed on to the server, and the results can be cached.

If the control is present, and if the cacheResults flag is FALSE (or not present, that is, an empty SEQUENCE), the client must bypass the cache, send the request to the server, and bypass adding the results to the cache.

If the control is present, and if the cacheResults flag is TRUE, then whether the search request is cached, the search is passed onto the server, and the results can be cached.

Notes:

1. The cacheResults must be a BER-encoded sequence. For coding convenience, the ldap.h include file defines the BER_ENCODED_BOOLEAN_TRUE and BER_ENCODED_BOOLEAN_FALSE constants. Additionally, the following constants are defined and represent the numeric OID for this control:
   - IBM_SERVER_HANDLED_SEARCH_REQUEST_OID
   - IBM_SERVER_HANDLED_SEARCH_REQUEST_OID_UTF8
2. This control is only supported by LDAP search operations.
3. This control is only applicable if client-side caching is enabled.

ibm-saslBindDigestUserName

Name: ibm-saslBindDigestUserName

Numeric OID:

1.3.18.0.2.10.13

Purpose:

Provides the ability to specify the user name authentication identity for a DIGEST-MD5 SASL authentication bind.

Criticality:

TRUE or FALSE. If TRUE, operations that do not support this control fail with LDAP_UNAVAILABLE_CRITICAL_EXTENSION. If FALSE, operations that do not support this control ignore its presence and process the request.

Value: A character string representing the user name. The string is in UTF-8 or the local EBCDIC code page as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle associated with the request. The ldctl_value.bv_val field contains the address of the string and the ldctl_value.bv_len field contains the length of the string (excluding the string delimiter).

Example:

The following is an example of defining an ibm-saslBindDigestUserName control.

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

Meaning:
If the control is present and DIGEST-MD5 authentication is specified, the user name is the identity used for authentication binding.

Notes:

1. For coding convenience, the `ldap.h` include file defines the `IBM_CLIENT_MD5_USER_NAME_OID`, `IBM_CLIENT_MD5_USER_NAME_OID_UTF8`, `IBM_CLIENT_DIGEST_USER_NAME_OID`, and `IBM_CLIENT_DIGEST_USER_NAME_OID_UTF8` constants for the numeric OID for this control.
2. This control is supported only by LDAP bind operations.

ibm-saslBindCramUserName

Name: `ibm-saslBindCramUserName`

Numeric OID:
`1.3.18.0.2.10.13`

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

Criticality:
`TRUE` or `FALSE`. If `TRUE`, operations that do not support this control fail with `LDAP_UNAVAILABLE_CRITICAL_EXTENSION`. If `FALSE`, operations that do not support this control ignore its presence and process the request.

Value: A character string representing the user name. The string is in UTF-8 or the local EBCDIC code page as determined by the `LDAP_OPT_UTF8_IO` option for the LDAP handle associated with the request. The `ldctl_value.bv_val` field contains the address of the string and the `ldctl_value.bv_len` field contains the length of the string (excluding the string delimiter). The user name must consist of characters that can be represented in the ISO8859-1 code page and must not contain any blanks.

Example:
The following is an example of defining an `ibm-saslBindCramUserName` control.

```
static LDAPControl userControl = {
     IBM_CLIENT_MD5_USER_NAME_OID,            /* OID */
     { 4, "juan" },                           /* username */
     LDAP_OPT_OFF                             /* non-critical */
};
```

Meaning:
If the control is present and CRAM-MD5 authentication is specified, the user name is the identity used for authentication binding.

Notes:

1. For coding convenience, the `ldap.h` include file defines the `IBM_CLIENT_MD5_USER_NAME_OID`, `IBM_CLIENT_MD5_USER_NAME_OID_UTF8`, `IBM_CLIENT_CRAM_USER_NAME_OID`, and `IBM_CLIENT_CRAM_USER_NAME_OID_UTF8` constants for the numeric OID for this control.

2. This control is supported only by LDAP bind operations.

ibm-saslBindDigestRealmName

Name: ibm-saslBindDigestRealmName

Numeric OID:
1.3.18.0.2.10.12

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

Criticality:
TRUE or FALSE. If TRUE, operations that do not support this control fail with LDAP_UNAVAILABLE_CRITICAL_EXTENSION. If FALSE, operations that do not support this control ignore its presence and process the request.

Value: A character string representing the realm name. The string is in UTF-8 or the local EBCDIC code page as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle associated with the request. The ldctl_value.bv_val field contains the address of the string and the ldctl_value.bv_len field contains the length of the string (excluding the string delimiter).

Example:
The following is an example of defining an ibm-saslBindDigestRealmName control.

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

Meaning:
If the control is present and DIGEST-MD5 authentication is specified, the realm name is used to select a realm in which to bind.

Notes:

1. For coding convenience, the ldap.h include file defines the IBM_CLIENT_MD5_REALM_NAME_OID, IBM_CLIENT_MD5_REALM_NAME_OID_UTF8, IBM_CLIENT_DIGEST_REALM_NAME_OID, and IBM_CLIENT_DIGEST_REALM_NAME_OID_UTF8 constants for the numeric OID for this control.
2. This control is supported only by LDAP bind operations.

Using RACF data

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

Thread safety

The LDAP programming interface is thread safe. Thread safety is 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 suggested as it causes multiple open TCP/IP socket connections between the client program and the directory server.

Client-side search results caching

Client-side search result caching is supported. It can be enabled for specific LDAP connections or globally for all connections. The **ldap_memcache_init()** and **ldap_memcache_set()** routines are used to specify search result caching for specific LDAP connections. The `LDAP_CLIENT_CACHE`, `LDAP_CLIENT_CACHE_MAX_SIZE`, and `LDAP_CLIENT_CACHE_TTL` environment variables are used to specify global search result caching. (See "LDAP client environment variables" on page 235 for details.) The `ibm-serverHandledSearchRequest` client control is used to disable search result caching for a specific search request.

When caching search results or retrieving the results of a previous search request, a case-sensitive comparison is performed between the base distinguished name in the search request and the base distinguished name in the cache. The distinguished names must be identical, including case, any white space characters, or escape sequences.

The results for a search request are added to the cache if the following conditions are true:

- The result code is `LDAP_SUCCESS`, `LDAP_REFERRAL`, or `LDAP_PARTIAL_RESULTS`.
- The search base distinguished name is included in the list of distinguished names for the cache.

A search request is satisfied from the cache if the following conditions are true:

- The LDAP server host name and port number must be the same.
- The bind mechanism and bind identity must be the same.
- The search parameters and search options must be the same.
- The `LDAP_OPT_REFERRALS`, `LDAP_OPT_REFHOPLIMIT`, `LDAP_OPT_PROTOCOL_VERSION`, `LDAP_OPT_REBIND_FN`, and `LDAP_OPT_EXT_REBIND_FN` options must be the same.

Restriction: The LDAP client cannot determine whether the contents of the cache are current. The application must make this determination. If the contents are not current (they are *stale*), the application should clear the cache.

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 beneficial in allowing multiple outstanding search operations from the client program. Searches that take less time to complete can be returned without waiting for a more complicated search to complete. However, there is some interplay with the thread safety support. 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 is held up until the **ldap_result()** on the first thread is complete. 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 that are possibly initiated on other threads from being started.

Guideline: When running in a multi-threaded environment, calls to **ldap_result()** should be made to wait for the first available result instead of waiting for specific results.

With synchronous operations, even though multiple operations can be initiated on separate threads, the thread safety support serializes 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 might be making calls to the LDAP programming interface.

Calling the LDAP APIs from other languages

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

LDAP client for Java

An industry-standard Java™ programming language interface exists to access the LDAP server directory services through the Java Naming and Directory Interface (JNDI). You can find the information about how to use the LDAP service provider interface (LDAP SPI) for JNDI in documentation from Oracle.

The JNDI is shipped as part of Java on z/OS. Use the JNDI that is shipped with Java and supported on z/OS.

Chapter 2. LDAP routines

This topic describes the Lightweight Directory Access Protocol (LDAP) application programming routines. These routines provide access through TCP/IP to directory services that accept the LDAP protocol.

The following deprecated routines are supported but have been replaced by newer, current LDAP routines. For detailed descriptions of these routines, see Chapter 3, "Deprecated LDAP routines," on page 221.

Guideline: Avoid using deprecated routines. Use current replacement routines instead.

Deprecated routine	Replacement routine
ldap_bind()	**ldap_simple_bind()**
ldap_bind_s()	**ldap_simple_bind_s()**
ldap_modrdn()	**ldap_rename()**
ldap_modrdn_s()	**ldap_rename_s()**
ldap_open()	**ldap_init or ldap_ssl_init()**
ldap_perror()	**ldap_parse_result()** or **ldap_get_errno()**
ldap_result2error()	**ldap_parse_result()**
ldap_ssl_start()	**ldap_ssl_client_init()** and **ldap_ssl_init()**

Table 1 lists current LDAP routines and the function each performs.

Table 1. Summary of the current LDAP routines

Name of routine	Function performed	For a detailed description, see ...
ldap_abandon()	Abandon an operation	"ldap_abandon(), ldap_abandon_ext()" on page 30
ldap_abandon_ext()	Abandon an operation	"ldap_abandon(), ldap_abandon_ext()" on page 30
ldap_add()	Add an entry to the LDAP directory	"ldap_add(), ldap_add_s(), ldap_add_ext(), ldap_add_ext_s()" on page 32
ldap_add_s()	Add an entry to the LDAP directory	"ldap_add(), ldap_add_s(), ldap_add_ext(), ldap_add_ext_s()" on page 32
ldap_add_control()	Create a control and insert it into a list of controls	"ldap_add_control()" on page 36
ldap_add_ext()	Add an entry to the LDAP directory	"ldap_add(), ldap_add_s(), ldap_add_ext(), ldap_add_ext_s()" on page 32
ldap_add_ext_s()	Add an entry to the LDAP directory	"ldap_add(), ldap_add_s(), ldap_add_ext(), ldap_add_ext_s()" on page 32
ldap_berfree_np()	Release storage for a binary value	"ldap_berfree_np()" on page 37
ldap_compare()	Compare an entry in the LDAP directory	"ldap_compare(), ldap_compare_s(), ldap_compare_ext(), ldap_compare_ext_s()" on page 38
ldap_compare_s()	Compare an entry in the LDAP directory	"ldap_compare(), ldap_compare_s(), ldap_compare_ext(), ldap_compare_ext_s()" on page 38

Table 1. Summary of the current LDAP routines (continued)

Name of routine	Function performed	For a detailed description, see ...
ldap_compare_ext()	Compare an entry in the LDAP directory	"ldap_compare(), ldap_compare_s(), ldap_compare_ext(), ldap_compare_ext_s()" on page 38
ldap_compare_ext_s()	Compare an entry in the LDAP directory	"ldap_compare(), ldap_compare_s(), ldap_compare_ext(), ldap_compare_ext_s()" on page 38
ldap_control_free()	Release the storage for an LDAP control	"ldap_control_free()" on page 42
ldap_controls_free()	Release the storage for an array of LDAP controls	"ldap_controls_free()" on page 43
ldap_convert_local_np()	Convert a text string from the local EBCDIC code page to UTF-8	"ldap_convert_local_np()" on page 44
ldap_convert_utf8_np()	Convert a text string from UTF-8 to the local EBCDIC code page	"ldap_convert_utf8_np()" on page 45
ldap_count_attributes()	Return the number of attributes in an LDAP search entry	"ldap_count_attributes()" on page 46
ldap_count_entries()	Return the number of search entries in an LDAP result	"ldap_count_entries()" on page 47
ldap_count_messages()	Return the number of messages in an LDAP result	"ldap_count_messages()" on page 48
ldap_count_references()	Return the number of search references in an LDAP result	"ldap_count_references()" on page 49
ldap_count_values()	Return the number of elements in an array of character strings	"ldap_count_values()" on page 50
ldap_count_values_len()	Return the number of elements in an array of binary values	"ldap_count_values_len()" on page 51
ldap_create_page_control()	Create a paged result control for use with an LDAP search request	"ldap_create_page_control()" on page 52
ldap_create_ persistentsearch_ control()	Create a persistent search control for use with an LDAP search request	**"ldap_create_persistentsearch_control()" on page 55**
ldap_create_sort_control()	Create a sort result request control for use with an LDAP search request	"ldap_create_sort_control()" on page 57
ldap_create_sort_keylist()	Create a list of sort keys	"ldap_create_sort_keylist()" on page 59
ldap_delete()	Delete an entry from the LDAP directory	"ldap_delete(), ldap_delete_s(), ldap_delete_ext(), ldap_delete_ext_s()" on page 61
ldap_delete_s()	Delete an entry from the LDAP directory	"ldap_delete(), ldap_delete_s(), ldap_delete_ext(), ldap_delete_ext_s()" on page 61

Table 1. Summary of the current LDAP routines (continued)

Name of routine	Function performed	For a detailed description, see ...
ldap_delete_ext()	Delete an entry from the LDAP directory	"ldap_delete(), ldap_delete_s(), ldap_delete_ext(), ldap_delete_ext_s()" on page 61
ldap_delete_ext_s()	Delete an entry from the LDAP directory	"ldap_delete(), ldap_delete_s(), ldap_delete_ext(), ldap_delete_ext_s()" on page 61
ldap_dn2ufn()	Parse a distinguished name and return a user-friendly name	"ldap_dn2ufn()" on page 64
ldap_dn2ufn_np()	Parse a distinguished name and return a user-friendly name	"ldap_dn2ufn_np()" on page 65
ldap_enetwork_domain_ get()	Return the eNetwork domain for the current user	"ldap_enetwork_domain_get()" on page 66
ldap_enetwork_domain_ set()	Set the eNetwork domain for the current user	"ldap_enetwork_domain_set()" on page 68
ldap_err2string()	Return a descriptive text message for an LDAP error code	"ldap_err2string()" on page 70
ldap_explode_dn()	Parse a distinguished name into an array of relative distinguished names	"ldap_explode_dn()" on page 71
ldap_explode_dn_np()	Parse a distinguished name and return an LDAP DN description	"ldap_explode_dn_np()" on page 73
ldap_explode_rdn()	Parse a relative distinguished name into an array of attributes	"ldap_explode_rdn()" on page 75
ldap_extended_operation()	Perform extended operations	"ldap_extended_operation(), ldap_extended_operation_s()" on page 76
ldap_extended_operation_s()	Perform extended operations	"ldap_extended_operation(), ldap_extended_operation_s()" on page 76
ldap_first_attribute()	Return the attribute type for the first attribute in an LDAP search entry	"ldap_first_attribute()" on page 79
ldap_first_entry()	Return the first search entry in an LDAP result	"ldap_first_entry()" on page 81
ldap_first_message()	Return the first message in an LDAP result	"ldap_first_message()" on page 82
ldap_first_reference()	Return the first search reference in an LDAP result	"ldap_first_reference()" on page 83
ldap_free_dndesc_np()	Release storage allocated for an LDAP DN description	"ldap_free_dndesc_np()" on page 84
ldap_free_sort_keylist()	Release storage allocated for a list of sort keys	"ldap_free_sort_keylist()" on page 85

Table 1. Summary of the current LDAP routines (continued)

Name of routine	Function performed	For a detailed description, see ...
ldap_free_urldesc()	Release storage allocated for an LDAP URL description	"ldap_free_urldesc()" on page 86
ldap_get_dn()	Return the distinguished name from a search entry	"ldap_get_dn()" on page 87
ldap_get_entry_controls_ np()	Return the server controls from a search entry message	"ldap_get_entry_controls_np()" on page 88
ldap_get_errno()	Return the last error code for an LDAP handle	"ldap_get_errno()" on page 89
ldap_get_function_vector()	Obtain the address of the LDAP function vector	"ldap_get_function_vector()" on page 90
ldap_get_lderrno()	Return information for the most recent error	"ldap_get_lderrno()" on page 92
ldap_get_option()	Return the value for an LDAP option	"ldap_get_option()" on page 93
ldap_get_values()	Return the attribute values as an array of character strings	"ldap_get_values()" on page 105
ldap_get_values_len()	Return the attribute values as an array of binary values	"ldap_get_values_len()" on page 106
ldap_init()	Create and initialize an LDAP handle for an SSL or non-SSL connection	"ldap_init()" on page 107
ldap_insert_control()	Insert an existing control into a list of controls	"ldap_insert_control()" on page 110
ldap_is_ldap_url()	Determine if a URL appears to be an LDAP URL	"ldap_is_ldap_url()" on page 111
ldap_is_ldap_url_np()	Determine if a URL appears to be an LDAP URL	"ldap_is_ldap_url_np()" on page 112
ldap_memcache_destroy()	Destroy a search result cache	"ldap_memcache_destroy()" on page 113
ldap_memcache_flush()	Remove entries from a search result cache	"ldap_memcache_flush()" on page 114
ldap_memcache_get()	Return the search result cache for an LDAP handle	"ldap_memcache_get()" on page 116
ldap_memcache_init()	Create a search result cache	"ldap_memcache_init()" on page 117
ldap_memcache_set()	Set the search result cache for an LDAP handle	"ldap_memcache_set()" on page 119
ldap_memcache_update()	Remove expired search result cache entries	"ldap_memcache_update()" on page 120
ldap_memfree()	Release storage allocated by the LDAP run time	"ldap_memfree()" on page 121

Table 1. Summary of the current LDAP routines (continued)

Name of routine	Function performed	For a detailed description, see ...
ldap_modify()	Modify an existing entry in the LDAP directory	"ldap_modify(), ldap_modify_s(), ldap_modify_ext(), ldap_modify_ext_s()" on page 122
ldap_modify_s()	Modify an existing entry in the LDAP directory	"ldap_modify(), ldap_modify_s(), ldap_modify_ext(), ldap_modify_ext_s()" on page 122
ldap_modify_ext()	Modify an existing entry in the LDAP directory	"ldap_modify(), ldap_modify_s(), ldap_modify_ext(), ldap_modify_ext_s()" on page 122
ldap_modify_ext_s()	Modify an existing entry in the LDAP directory	"ldap_modify(), ldap_modify_s(), ldap_modify_ext(), ldap_modify_ext_s()" on page 122
ldap_mods_free()	Release storage allocated for an array of attribute modifications	"ldap_mods_free()" on page 126
ldap_msgfree()	Release storage for an LDAP message	"ldap_msgfree()" on page 127
ldap_msgid()	Return the message identifier	"ldap_msgid()" on page 128
ldap_msgtype()	Return the message type	"ldap_msgtype()" on page 129
ldap_next_attribute()	Return the attribute type for the next attribute in an LDAP search entry	"ldap_next_attribute()" on page 130
ldap_next_entry()	Return the next search entry in an LDAP result	"ldap_next_entry()" on page 131
ldap_next_message()	Return the next LDAP message in an LDAP result	"ldap_next_message()" on page 132
ldap_next_reference()	Return the next search reference in an LDAP result	"ldap_next_reference()" on page 133
ldap_parse_entrychange_control()	Parse an entry change notification server control returned in an LDAP search response	"ldap_parse_entrychange_control()" on page 134
ldap_parse_extended_result()	Parse an LDAP extended result message	"ldap_parse_extended_result()" on page 136
ldap_parse_page_control()	Parse a paged results server control returned in an LDAP search response	"ldap_parse_page_control()" on page 138
ldap_parse_pwdpolicy_response()	Parse a password policy control response returned in an LDAP message	"ldap_parse_pwdpolicy_response()" on page 140
ldap_parse_reference_np()	Parse an LDAP search continuation reference message	"ldap_parse_reference_np()" on page 142
ldap_parse_result()	Parse an LDAP result message	"ldap_parse_result()" on page 144

Table 1. Summary of the current LDAP routines (continued)

Name of routine	Function performed	For a detailed description, see ...
ldap_parse_sasl_bind_result()	Parse an LDAP SASL bind result message	"ldap_parse_sasl_bind_result()" on page 146
ldap_parse_sort_control()	Parse a sort results response control returned in an LDAP search response	"ldap_parse_sort_control()" on page 147
ldap_pwdpolicy_err2string	Return a descriptive text message for an LDAP password policy control response error or warning code	"ldap_pwdpolicy_err2string()" on page 149
ldap_remove_control()	Remove a control from a list of controls	"ldap_remove_control()" on page 150
ldap_rename()	Rename an entry in the LDAP directory	"ldap_rename(), ldap_rename_s()" on page 151
ldap_rename_s()	Rename an entry in the LDAP directory	"ldap_rename(), ldap_rename_s()" on page 151
ldap_result()	Return the result message for an LDAP request	"ldap_result()" on page 154
ldap_sasl_bind()	Bind to the LDAP server using the Simple Authentication and Security Layer	"ldap_sasl_bind(), ldap_sasl_bind_s()" on page 157
ldap_sasl_bind_s()	Bind to the LDAP server using the Simple Authentication and Security Layer	"ldap_sasl_bind(), ldap_sasl_bind_s()" on page 157
ldap_search()	Search the LDAP directory	"ldap_search(), ldap_search_s(), ldap_search_st(), ldap_search_ext(), ldap_search_ext_s()" on page 164
ldap_search_s()	Search the LDAP directory	"ldap_search(), ldap_search_s(), ldap_search_st(), ldap_search_ext(), ldap_search_ext_s()" on page 164
ldap_search_st()	Search the LDAP directory	"ldap_search(), ldap_search_s(), ldap_search_st(), ldap_search_ext(), ldap_search_ext_s()" on page 164
ldap_search_ext()	Search the LDAP directory	"ldap_search(), ldap_search_s(), ldap_search_st(), ldap_search_ext(), ldap_search_ext_s()" on page 164
ldap_search_ext_s()	Search the LDAP directory	"ldap_search(), ldap_search_s(), ldap_search_st(), ldap_search_ext(), ldap_search_ext_s()" on page 164
ldap_server_conf_save()	Save the LDAP server information list	"ldap_server_conf_save()" on page 172
ldap_server_free_list()	Release the LDAP server information list	"ldap_server_free_list()" on page 174
ldap_server_locate()	Locate the LDAP servers	"ldap_server_locate()" on page 175
ldap_set_option()	Set the value for an LDAP option	"ldap_set_option(), ldap_set_option_np()" on page 182

Table 1. Summary of the current LDAP routines (continued)

Name of routine	Function performed	For a detailed description, see ...
ldap_set_option_np()	Set the value for an LDAP option	"ldap_set_option(), ldap_set_option_np()" on page 182
ldap_set_rebind_proc()	Specify the routine to be called when binding to another LDAP server	"ldap_set_rebind_proc()" on page 195
ldap_simple_bind()	Bind to the LDAP server using a distinguished name (DN) and password	"ldap_simple_bind(), ldap_simple_bind_s()" on page 196
ldap_simple_bind_s()	Bind to the LDAP server using a distinguished name (DN) and password	"ldap_simple_bind(), ldap_simple_bind_s()" on page 196
ldap_ssl_client_init()	Initialize the SSL client run time	"ldap_ssl_client_init()" on page 198
ldap_ssl_init()	Create and initialize an LDAP handle for an SSL connection	"ldap_ssl_init()" on page 200
ldap_start_tls_s_np()	Start TLS for a connection	"ldap_start_tls_s_np()" on page 203
ldap_stop_tls_s_np()	Stop TLS for a connection	"ldap_stop_tls_s_np()" on page 205
ldap_unbind()	Close the connection to the LDAP server and release the LDAP handle	"ldap_unbind(), ldap_unbind_s()" on page 206
ldap_unbind_s()	Close the connection to the LDAP server and release the LDAP handle	"ldap_unbind(), ldap_unbind_s()" on page 206
ldap_url_parse()	Parse an LDAP URL	"ldap_url_parse()" on page 207
ldap_url_parse_np()	Parse an LDAP URL	"ldap_url_parse_np()" on page 210
ldap_url_search()	Search the LDAP directory using an LDAP URL	"ldap_url_search(), ldap_url_search_s(), ldap_url_search_st()" on page 212
ldap_url_search_s()	Search the LDAP directory using an LDAP URL	"ldap_url_search(), ldap_url_search_s(), ldap_url_search_st()" on page 212
ldap_url_search_st()	Search the LDAP directory using an LDAP URL	"ldap_url_search(), ldap_url_search_s(), ldap_url_search_st()" on page 212
ldap_value_free()	Release storage allocated for an array of character strings	"ldap_value_free()" on page 216
ldap_value_free_len()	Release storage allocated for an array of binary values	"ldap_value_free_len()" on page 217
ldap_version()	Return LDAP version information	"ldap_version()" on page 218

ldap_abandon(), ldap_abandon_ext()

Purpose

Abandon an operation

Format

```
#include <ldap.h>

int ldap_abandon(
    LDAP *                  ld,
    int                     msgid)

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

Parameters

Input:

ld Specifies the LDAP handle.

msgid
Specifies the message identifier of a request that is still in progress.

serverctrls
Specifies an array of server controls for the abandon request. The end of the array is indicated by a NULL address. If NULL is specified for this parameter, the server controls specified by the LDAP_OPT_SERVER_CONTROLS option for the LDAP handle are used. If NULL is specified for this parameter and the LDAP_OPT_SERVER_CONTROLS option has not been set for the LDAP handle, no server controls are used. To override the server controls for the LDAP handle so that no controls are used, specify a server controls array consisting of a NULL address. (Control values for this routine vary depending on whether you are specifying server or client controls. See “LDAP controls” on page 15 for details.)

clientctrls
Specifies an array of client controls for the abandon request. The end of the array is indicated by a NULL address. If NULL is specified for this parameter, the client controls specified by the LDAP_OPT_CLIENT_CONTROLS option for the LDAP handle are used. If NULL is specified for this parameter and the LDAP_OPT_CLIENT_CONTROLS option has not been set for the LDAP handle, no client controls are used. To override the client controls for the LDAP handle so that no controls are used, specify a client controls array consisting of a NULL address. (Control values for this routine vary depending on whether you are specifying server or client controls. See “LDAP controls” on page 15 for details.)

Usage

The **ldap_abandon()** and **ldap_abandon_ext()** routines send an abandon request to the LDAP server for the request identified by the message identifier. The LDAP server cancels the request and does not return a result message. If the request has already completed, the result message is purged and is not returned to the application.

An application rebind exit routine cannot abandon the request causing the referral or any request in the referral chain. However, the application can abandon the request causing the referral if a timeout occurs and control is returned to the application by the **ldap_result()** routine. Abandoning the original referral request causes all requests in the referral chain to be abandoned.

Client controls specified by the LDAP_OPT_CLIENT_CONTROLS option and server controls specified by the LDAP_OPT_SERVER_CONTROLS option are used by the **ldap_abandon()** routine. These controls are also used by the **ldap_abandon_ext()** routine unless overridden by the serverctrls and clientctrls parameters.

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes listed in the ldap.h include file.

The following are some common errors for this routine:

LDAP_INVALID_STATE
An unbind is in progress or the requested message is currently being processed by the LDAP run time.

LDAP_NO_MATCHING_REQUEST
The message identifier does not refer to an outstanding request.

LDAP_NO_MEMORY
Insufficient storage available.

LDAP_NOT_SUPPORTED
LDAP protocol version 3 is required to specify server or client controls.

LDAP_PARAM_ERROR
A parameter is not valid.

LDAP_SERVER_DOWN
Unable to send request to server.

ldap_add(), ldap_add_s(), ldap_add_ext(), ldap_add_ext_s()

Purpose

Add an entry to the LDAP directory.

Format

```
#include <ldap.h>

typedef struct ldapmod {
    int                         mod_op;
    char *                      mod_type;
    union {
        char **                 modv_strvals;
        BerVal **               modv_bvals;
    } mod_vals;
    struct ldapmod *            mod_next;
} LDAPMod;

#define LDAP_MOD_BVALUES        0x80

#define mod_values              mod_vals.modv_strvals
#define mod_bvalues             mod_vals.modv_bvals

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

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

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

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

Parameters

Input

ld Specifies the LDAP handle.

dn Specifies the distinguished name for the directory entry as a null-terminated character string in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. A zero-length name is not allowed for an add request.

mods
Specifies the attributes for the directory entry. The mod_op field is ignored other than checking the LDAP_MOD_BVALUES flag. The mod_type field specifies the attribute type as a null-terminated character string in UTF-8 or the local

EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. The modv_strvals field can be used for character values and the modv_bvals field can be used for binary values. The supplied values are in binary if the LDAP_MOD_BVALUES flag is set. Otherwise, the supplied values are null-terminated character strings in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle.

serverctrls
Specifies an array of server controls for the add request. The end of the array is indicated by a NULL address. If NULL is specified for this parameter, the server controls specified by the LDAP_OPT_SERVER_CONTROLS option for the LDAP handle are used. If NULL is specified for this parameter and the LDAP_OPT_SERVER_CONTROLS option has not been set for the LDAP handle, no server controls are used. To override the server controls for the LDAP handle so that no controls are used, specify a server controls array consisting of a NULL address. (Control values for this routine vary depending on whether you are specifying server or client controls. See "LDAP controls" on page 15 for details.)

clientctrls
Specifies an array of client controls for the add request. The end of the array is indicated by a NULL address. If NULL is specified for this parameter, the client controls specified by the LDAP_OPT_CLIENT_CONTROLS option for the LDAP handle are used. If NULL is specified for this parameter and the LDAP_OPT_CLIENT_CONTROLS option has not been set for the LDAP handle, no client controls are used. To override the client controls for the LDAP handle so that no controls are used, specify a client controls array consisting of a NULL address. (Control values for this routine vary depending on whether you are specifying server or client controls. See "LDAP controls" on page 15 for details.)

Output

msgidp
Returns the message identifier assigned to the add request message. The application can use this value when calling the **ldap_result()** routine to wait for the add result message.

Usage

The **ldap_add()** and **ldap_add_ext()** routines send the request to the LDAP server and return control to the application. The application must call the **ldap_result()** routine to obtain the result.

The **ldap_add_s()** and **ldap_add_ext_s()** routines send the request to the LDAP server and wait for the completion of the request. The add request is abandoned if the client is unable to wait for the response because of an error from the **ldap_result()** routine.

The parent entry must exist. For example, if an entry named "cn=John Doe, ou=Manufacturing, o=Acme" is being added, the entry named "ou=Manufacturing, o=Acme" must exist.

The supplied attributes should include all attributes making up the low-level RDN (relative distinguished name) of the entry name and the objectClass attribute and any mandatory attributes for the specified object classes. Each attribute must have

at least one value. The maximum value length is 2147483647. The z/OS LDAP server adds any missing RDN attributes and values if the addition does not cause an object class or constraint violation.

Client controls specified by the LDAP_OPT_CLIENT_CONTROLS option and server controls specified by the LDAP_OPT_SERVER_CONTROLS option are used by the **ldap_add()** and **ldap_add_s()** routines. These controls are also used by the **ldap_add_ext()** and **ldap_add_ext_s()** routines unless overridden by the serverctrls and clientctrls parameters.

Function return value

The **ldap_add()** routine returns -1 if a client error is detected. Otherwise, it returns the message identifier assigned to the add request. If the return value is -1, the application should call the **ldap_get_errno()** routine to get the error code. Errors reported by the LDAP server are not returned by the **ldap_add()** routine. Instead, the application must call the **ldap_parse_result()** routine to obtain the result code from the result message returned by the **ldap_result()** routine.

The **ldap_add_ext()** routine returns LDAP_SUCCESS if the request is sent to the LDAP server. Otherwise, the return value is one of the error codes listed in the ldap.h include file. Errors reported by the LDAP server are not returned by the **ldap_add_ext()** routine. Instead, the application must call the **ldap_parse_result()** routine to obtain the result code from the result message returned by the **ldap_result()** routine.

The **ldap_add_s()** and **ldap_add_ext_s()** routines return LDAP_SUCCESS if the request is successful. Otherwise, the return value is one of the error codes listed in the ldap.h include file. The return value includes errors detected by the LDAP client and errors detected by the LDAP server.

The following are some common client errors:

LDAP_INVALID_STATE
An unbind request has been issued for the LDAP handle.

LDAP_LOCAL_ERROR
A system function reported an error.

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_NOT_SUPPORTED
The LDAP protocol version must be LDAP_VERSION3 to specify server or client controls.

LDAP_PARAM_ERROR
A parameter is not valid.

LDAP_SERVER_DOWN
Network connection failed.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION
A critical client control is either not recognized or is not supported for a add operation.

The following are some common server result codes:

LDAP_ALREADY_EXISTS
The entry exists.

LDAP_INSUFFICIENT_ACCESS
Not authorized to add entry.

LDAP_NO_SUCH_OBJECT
The parent entry does not exist.

LDAP_OBJECT_CLASS_VIOLATION
Either a mandatory attribute is not included or an attribute is not allowed by the object class definition.

LDAP_REFERRAL
The parent entry is not in the current LDAP server.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION
A critical server control is either not recognized or is not supported for a add operation.

LDAP_UNDEFINED_TYPE
An attribute type is not defined in the directory schema.

ldap_add_control()

Purpose

Create a control and insert it into a list of controls

Format

```
#include <ldap.h>

int ldap_add_control(
    char *                  oid,
    ber_len_t               len,
    char *                  value,
    int                     is_critical,
    LDAPControl ***         control_list)
```

Parameters

Input

oid Specifies the control type, represented as a string.

len Specifies the length of the value string.

value Specifies the data associated with the control.

is_critical
Specify 1 if this is a critical control, otherwise specify 0.

Output

control_list
Specifies the address of the control list. A new control list is created if there is no control list. (The location pointed to by the *control_list* parameter contains NULL.) Otherwise, the existing control list is expanded and the new control is added to the list. The **ldap_controls_free()** routine should be called to release the controls when they are no longer needed.

Usage

The **ldap_add_control()** routine creates a control (using the oid, len, value, and is_critical values) and inserts it into a list of controls specified by control_list.

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, the return value is one of the LDAP error codes listed in the ldap.h include file.

The following are some common client errors:

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_berfree_np()

Purpose

Release storage for a binary value

Format

```
#include <ldap.h>

void ldap_berfree_np(
    BerVal *                  val)
```

Parameters

Input

val
Specifies the binary value to free.

Usage

The **ldap_berfree_np()** routine releases the storage allocated for a binary value. The `BerVal` structure and the binary value are freed.

Function return value

There is no function return value.

ldap_compare(), ldap_compare_s(), ldap_compare_ext(), ldap_compare_ext_s()

Purpose

Compare an attribute value to an attribute value for an entry in the LDAP directory

Format

```
#include <ldap.h>

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

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

int ldap_compare_ext(
    LDAP *                  ld,
    const char *            dn,
    const char *            attr,
    BerVal *                bvalue,
    LDAPControl *           serverctrls[],
    LDAPControl *           clientctrls[],
    int *                   msgidp)

int ldap_compare_ext_s(
    LDAP *                  ld,
    const char *            dn,
    const char *            attr,
    BerVal *                bvalue,
    LDAPControl *           serverctrls[],
    LDAPControl *           clientctrls[])
```

Parameters

Input

ld Specifies the LDAP handle.

dn Specifies the distinguished name for the directory entry as a null-terminated character string in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. A zero-length name can be used to specify the root DSE.

attr
Specifies the attribute type as a null-terminated character string in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle.

value
Specifies the attribute value as a null-terminated character string in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle.

bvalue
Specifies the attribute value as a binary octet string. The value is sent to the LDAP server without any conversion. The maximum value length is 2147483647.

serverctrls
Specifies an array of server controls for the compare request. The end of the array is indicated by a NULL address. If NULL is specified for this parameter, the server controls specified by the LDAP_OPT_SERVER_CONTROLS option for the LDAP handle are used. If NULL is specified for this parameter and the LDAP_OPT_SERVER_CONTROLS option has not been set for the LDAP handle, no server controls are used. To override the server controls for the LDAP handle so that no controls are used, specify a server controls array consisting of a NULL address. (Control values for this routine vary depending on whether you are specifying server or client controls. See "LDAP controls" on page 15 for details.)

clientctrls
Specifies an array of client controls for the compare request. The end of the array is indicated by a NULL address. If NULL is specified for this parameter, the client controls specified by the LDAP_OPT_CLIENT_CONTROLS option for the LDAP handle are used. If NULL is specified for this parameter and the LDAP_OPT_CLIENT_CONTROLS option has not been set for the LDAP handle, no client controls are used. To override the client controls for the LDAP handle so that no controls are used, specify a client controls array consisting of a NULL address. (Control values for this routine vary depending on whether you are specifying server or client controls. See "LDAP controls" on page 15 for details.)

Output

msgidp
Returns the message identifier assigned to the compare request message. The application can use this value when calling the **ldap_result()** routine to wait for the compare result message.

Usage

The **ldap_compare()** and **ldap_compare_ext()** routines send the request to the LDAP server and return control to the application. The application must call the **ldap_result()** routine to obtain the result.

The **ldap_compare_s()** and **ldap_compare_ext_s()** routines send the request to the LDAP server and wait for the completion of the request. The compare request is abandoned if the client is unable to wait for the response because of an error from the **ldap_result()** routine.

The supplied attribute value is compared to the attribute value for the directory entry. The result code is LDAP_COMPARE_TRUE if the values are the same and LDAP_COMPARE_FALSE if the values are not the same. Any other result code indicates an error.

Client controls specified by the LDAP_OPT_CLIENT_CONTROLS and server controls specified by the LDAP_OPT_SERVER_CONTROLS options are used by the **ldap_compare()** and **ldap_compare_s()** routines. These controls are also used by the **ldap_compare_ext()** and **ldap_compare_ext_s()** routines unless overridden by the serverctrls and clientctrls parameters.

Function return value

The **ldap_compare()** routine returns -1 if a client error is detected. Otherwise, it returns the message identifier assigned to the compare request. If the return value is -1, the application should call the **ldap_get_errno()** routine to get the error code. Errors reported by the LDAP server are not returned by the **ldap_compare()** routine. Instead, the application must call the **ldap_parse_result()** routine to obtain the result code from the result message returned by the **ldap_result()** routine.

The **ldap_compare_ext()** routine returns LDAP_SUCCESS if the request is sent to the LDAP server. Otherwise, the return value is one of the error codes listed in the ldap.h include file. Errors reported by the LDAP server are not returned by the **ldap_compare_ext()** routine. Instead, the application must call the **ldap_parse_result()** routine to obtain the result code from the result message returned by the **ldap_result()** routine.

The **ldap_compare_s()** and **ldap_compare_ext_s()** routines return either LDAP_COMPARE_TRUE or LDAP_COMPARE_FALSE if the request is successful. Otherwise, the return value is one of the error codes listed in the ldap.h include file. The return value includes errors detected by the LDAP client and including errors detected by the LDAP server.

The following are some common client errors:

LDAP_INVALID_STATE
An unbind request has been issued for the LDAP handle.

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_NOT_SUPPORTED
The LDAP protocol version must be LDAP_VERSION3 to specify server or client controls.

LDAP_PARAM_ERROR
A parameter is not valid.

LDAP_SERVER_DOWN
Network connection failed.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION
A critical client control is either not recognized or is not supported for a compare operation.

The following are some common server result codes:

LDAP_COMPARE_FALSE
The attribute values are not the same.

LDAP_COMPARE_TRUE
The attribute values are the same.

LDAP_INSUFFICIENT_ACCESS
Not authorized to access the directory entry.

LDAP_NO_SUCH_ATTRIBUTE
The directory entry does not have the specified attribute.

LDAP_NO_SUCH_OBJECT
The directory entry does not exist.

LDAP_REFERRAL
The entry is not in the current LDAP server.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION
A critical server control is either not recognized or is not supported for a compare operation.

LDAP_UNDEFINED_TYPE
The attribute type is not defined in the directory schema.

ldap_control_free()

Purpose

Release the storage for an LDAP control

Format

```
#include <ldap.h>

void ldap_control_free(
    LDAPControl *              ctrl)
```

Parameters

Input

ctrl
Specifies the LDAP control.

Usage

The **ldap_control_free()** routine releases a single LDAP control. The `LDAPControl` structure is released along with the control data objects.

Function return value

There is no function return value.

ldap_controls_free()

Purpose

Release the storage for an array of LDAP controls

Format

```
#include <ldap.h>

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

Parameters

Input

ctrls
Specifies the array of LDAP controls. The end of the array is indicated by a `NULL` control address.

Usage

The **ldap_controls_free()** routine releases an array of LDAP controls. The address array and each `LDAPControl` structure are released along with the control data objects.

Function return value

There is no function return value.

ldap_convert_local_np()

Purpose

Convert a text string from the local EBCDIC code page to UTF-8

Format

```
#include <ldap.h>

int ldap_convert_local_np(
    LDAP *                          ld,
    const char *                    local_string,
    char **                         utf8_string)
```

Parameters

Input

ld Specifies the LDAP handle.

local_string
Specifies the string to be converted.

Output

utf8_string
Returns the converted string. The **ldap_memfree()** routine should be called to release the string when it is no longer needed.

Usage

The **ldap_convert_local_np()** routine converts a text string from the local EBCDIC code page to UTF-8. For more information about the conversion process, see the description of the **iconv()** routine in *z/OS XL C/C++ Runtime Library Reference*.

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes listed in the ldap.h include file.

The following are some common errors for this routine:

LDAP_BAD_STRING_ENCODING
The input string contains a character sequence that is not valid.

LDAP_LOCAL_ERROR
The **iconv()** routine failed.

LDAP_NO_MEMORY
Insufficient storage available.

ldap_convert_utf8_np()

Purpose

Convert a text string from UTF-8 to the local EBCDIC code page

Format

```
#include <ldap.h>

int ldap_convert_utf8_np(
    LDAP *                      ld,
    const char *                utf8_string,
    char **                     local_string)
```

Parameters

Input

ld Specifies the LDAP handle.

utf8_string
Specifies the string to be converted.

Output

local_string
Returns the converted string. The application should call the **ldap_memfree()** routine to release the string when it is no longer needed.

Usage

The **ldap_convert_utf8_np()** routine converts a text string from UTF-8 to the local EBCDIC code page. UTF-8 characters that cannot be represented in the local EBCDIC code page are replaced by a substitute character as determined by the local EBCDIC code page definition. For more information about the conversion process, see the description of the **iconv()** routine in *z/OS XL C/C++ Runtime Library Reference*.

Function return value

The function return value is `LDAP_SUCCESS` if no error is detected. Otherwise, it is one of the LDAP error codes listed in the `ldap.h` include file.

The following are some common errors for this routine:

LDAP_BAD_STRING_ENCODING
The input string contains a character sequence that is not valid.

LDAP_LOCAL_ERROR
The **iconv()** routine failed.

LDAP_NO_MEMORY
Insufficient storage available.

ldap_count_attributes()

Purpose

Return the number of attributes in an LDAP search entry

Format

```
#include <ldap.h>

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

Parameters

Input

ld Specifies the LDAP handle.

entry
Specifies the LDAP entry returned by the **ldap_first_entry()** or **ldap_next_entry()** routine.

Usage

The **ldap_count_attributes()** routine returns the number of attributes in an LDAP search entry.

Function return value

The function return value is the number of attributes in the entry or -1 if an error is detected. When the return value is -1, the application can call the **ldap_get_errno()** routine to get the error code.

The following are some common errors for this routine:

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_count_entries()

Purpose

Return the number of search entries in an LDAP result

Format

```
#include <ldap.h>

int ldap_count_entries(
    LDAP *                      ld,
    LDAPMessage *               msg)
```

Parameters

Input

ld Specifies the LDAP handle.

msg
Specifies the LDAP message.

Usage

The **ldap_count_entries()** routine returns the number of search entries in an LDAP result. The count includes the specified message and any messages chained to that message. This count is the total number of search entries in the result if the message is the result message returned by **ldap_result()** or one of the synchronous search request routines. The count is the number of search entries still to be processed if the message is returned by the **ldap_first_message()**, **ldap_next_message()**, **ldap_first_entry()**, **ldap_next_entry()**, **ldap_first_reference()**, or **ldap_next_reference()** routine.

Function return value

The function return value is the number of search entries or -1 if an error is detected. When the return value is -1, the application can call the **ldap_get_errno()** routine to get the error code.

The following are some common errors for this routine:

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_count_messages()

Purpose

Return the number of messages in an LDAP result

Format

```
#include <ldap.h>

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

Parameters

Input

ld Specifies the LDAP handle.

msg
Specifies the LDAP message.

Usage

The **ldap_count_messages()** routine returns the number of messages in an LDAP result. The count includes the specified message and any messages chained to that message. This count is the total number of messages in the result if the message is the result message returned by **ldap_result()** or one of the synchronous search request routines. The count is the number of messages still to be processed if the message is returned by the **ldap_first_message()**, **ldap_next_message()**, **ldap_first_entry()**, **ldap_next_entry()**, **ldap_first_reference()**, or **ldap_next_reference()** routine.

The **ldap_count_messages()** routine counts the number of messages without regard to the message type. Use the **ldap_count_entries()** or **ldap_count_references()** routine if you want to know the number of messages of a particular type.

Function return value

The function return value is the number of messages or -1 if an error is detected. When the return value is -1, the application can call the **ldap_get_errno()** routine to get the error code.

The following are some common errors for this routine:

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_count_references()

Purpose

Return the number of search references in an LDAP result

Format

```
#include <ldap.h>

int ldap_count_references(
    LDAP *                          ld,
    LDAPMessage *                   msg)
```

Parameters

Input

ld Specifies the LDAP handle.

msg
Specifies the LDAP message.

Usage

The **ldap_count_references()** routine returns the number of search references in an LDAP result. The count includes the specified message and any messages chained to that message. This count is the total number of search references in the result if the message is the result message returned by **ldap_result()** or one of the synchronous search request routines. The count is the number of search references still to be processed if the message is returned by the **ldap_first_message()**, **ldap_next_message()**, **ldap_first_entry()**, **ldap_next_entry()**, **ldap_first_reference()** or **ldap_next_reference()** routine.

Function return value

The function return value is the number of search references or -1 if an error is detected. When the return value is -1, the application can call the **ldap_get_errno()** routine to get the error code.

The following are some common errors for this routine:

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_count_values()

Purpose

Return the number of elements in an array of character strings

Format

```
#include <ldap.h>

int ldap_count_values(
    const char *              vals[])
```

Parameters

Input

vals
Specifies the array of character strings. The end of the array is indicated by a NULL address.

Usage

The **ldap_count_values()** routine returns the number of elements in an array of character strings.

Function return value

The function return value is the number of elements in the array.

ldap_count_values_len()

Purpose

Return the number of elements in an array of binary values

Format

```
#include <ldap.h>

int ldap_count_values_len(
    BerVal *                bvals[])
```

Parameters

Input

bvals
Specifies the array of binary values. The end of the array is indicated by a NULL address.

Usage

The **ldap_count_values_len()** routine returns the number of elements in an array of binary values.

Function return value

The function return value is the number of elements in the array.

ldap_create_page_control()

Purpose

Create a paged results control for use with an LDAP search request

Format

```
#include <ldap.h>

int ldap_create_page_control(
    LDAP *                ld,
    unsigned long         page_size,
    BerVal *              cookie,
    int                   is_critical,
    LDAPControl **        control)
```

Parameters

Input

ld Specifies the LDAP handle.

page_size
Specifies the page size. The maximum page size is 2147483647.

cookie
Specifies the cookie returned by the LDAP server for the previous request. Either specify NULL for this parameter or provide a zero-length cookie to create the initial paged results control.

is_critical
Specify 1 if this is a critical control, otherwise specify 0.

Output

control
Returns the paged results control. The **ldap_control_free()** routine should be called to release the control when it is no longer needed. The **ldap_insert_control()** routine can be used to add the control to a list of controls for input to the **ldap_search_ext()** or **ldap_search_ext_s()** routine.

Usage

RFC 2696: LDAP Control Extension for Simple Paged Results Manipulation provides paging capabilities for LDAP clients that receive a subset of search results (page) instead of the entire list. The next page of entries is returned to the client application for each subsequent paged results search request submitted by the client until the operation is canceled or the last result is returned. The server ignores the paged results control if the *page_size* is greater than or equal to the size limit value in the search request. A paged results control is not returned by the server in this case.

The **ldap_create_page_control()** routine takes as input a *page_size* and a *cookie* and builds a paged results server control (1.2.840.113556.1.4.319). The application then adds this control to the list of controls sent to the server on an LDAP search request to request that the server return the results in pages instead of all at once. This is done by using the **ldap_search_ext()** or **ldap_search_ext_s()** routine.

A zero-length *cookie* indicates this is the initial search request. The LDAP server returns a paged results control in the search response message if no errors are detected. The **ldap_parse_page_control()** routine can be used to parse the returned control and extract the total entry count and the new *cookie*. This *cookie* can be used to create the paged results control to retrieve the next page of results. Each search response returns the next page of results and a paged results control with a new cookie. A zero-length *cookie* is returned by the LDAP server with the final page of results. The search requests used to obtain the successive result pages must be the same as the original search request, other than the message identifier, *page_size* and *cookie*, or the server returns LDAP_UNWILLING_TO_PERFORM.

The client can cancel the remaining search results by sending a search request with the last *cookie* returned by the server and a *page_size* of zero. The remaining search results are discarded by the server and an LDAP search response is returned with a paged results control containing a zero-length cookie.

The LDAP server might limit the number of outstanding paged search results requests for a given client or for all clients. A server with a limit on the number of outstanding paged results requests returns LDAP_UNWILLING_TO_PERFORM in the search response message if either a new paged results search request cannot be started or an existing search request cannot be continued because the search results have been deleted.

There is no guarantee to the client application that the results of a search query remained unchanged through the life of a set of paged results request/response sequences. If the result set for the query changed since the initial search request specifying paged results, the client application might not receive all the entries matching the search criteria.

The client application must turn off automatic referral chasing and process referrals itself when issuing a paged search request. Otherwise, results may be incomplete, more entries than requested may be returned in a given page, or an error might occur attempting a paged search continuation. Automatic referral chasing is turned off by setting the LDAP_OPT_REFERRALS to LDAP_OPT_OFF using the **ldap_set_option()** or **ldap_set_option_np()** routine. When chasing referrals, the client application must send an initial paged results request, with a zero-length *cookie*, to each of the referral servers. The original LDAP server does not ensure that the referral server supports the paged results control. Multiple lists are returned to the client application, one by each referral server. It is the client application's decision as to how best to present this information to the user.

The sort results control can be used with the paged results control to apply the paging capability to a set of sorted results. The sort is performed on the entire set of search results before the first page is returned to the LDAP client. See “ldap_create_sort_control()” on page 57 for the description of the **ldap_create_sort_control()** routine for more information about obtaining sorted results.

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes listed in the ldap.h include file.

The following are some common client errors:

LDAP_NO_MEMORY
Insufficient storage is available

LDAP_PARAM_ERROR
A parameter is not valid

ldap_create_persistentsearch_control()

Purpose

Create a persistent search control for use with an LDAP search request

Format

```
#include <ldap.h>

int ldap_create_persistentsearch_control(
    LDAP *                      ld,
    int                         change_types,
    int                         changes_only,
    int                         return_echg_controls,
    int                         is_critical,
    LDAPControl **              control)
```

Parameters

Input

ld Specifies the LDAP handle.

change_types
Specifies the types of changes that should be included in the persistent search results. This is a bit-sensitive value that must be set to LDAP_CHANGETYPE_ANY or to any combination of the following values:

- LDAP_CHANGETYPE_ADD
- LDAP_CHANGETYPE_DELETE
- LDAP_CHANGETYPE_MODIFY
- LDAP_CHANGETYPE_MODDN.

changes_only
Specifies whether the search results should contain only changed entries. Specify 1 if only changed entries matching the search criteria should be returned. Specify 0 if all entries matching the search criteria should be returned before starting to monitor for changes.

return_echg_controls
Specifies whether the entry change notification control should be returned with each search result entry returned during the monitor phase of the search. Specify 1 if you want entry change notification controls, otherwise specify 0.

is_critical
Specify 1 if this is a critical control, otherwise specify 0.

Output

control
Returns the persistent search control. The **ldap_control_free()** routine should be called to release the control when it is no longer needed. The **ldap_insert_control()** routine can be used to add the control to a list of controls for input to the **ldap_search_ext()** routine.

Usage

The persistent search control provides the application with the ability to monitor changes to directory entries. This control should be used with asynchronous search

requests because the search does not complete until either the request is abandoned or the server connection is closed.

The **ldap_create_persistentsearch_control()** routine builds a persistent search server control (`2.16.840.1.113730.3.4.3`). The application then adds this control to the list of controls sent to the server on an LDAP search request. The `LDAP_OPT_DEREF` option should be set to `LDAP_DEREF_NEVER` or `LDAP_DEREF_FINDING` when issuing the search request.

Upon receiving the persistent search control, the LDAP server processes the search request as follows:

- Existing entries that match the search criteria are returned unless a nonzero value is specified for the `changes_only` parameter. The search results for existing entries do not contain the entry change notification server control.
- A search result completion message is not returned. Instead, the search operation remains active and monitors changes to the directory until the client abandons the request or an unbind is performed.
- As changes are made, the affected entries are returned to the client if they match the search criteria and if the operation causing the change is included in the list specified by the `change_types` parameter. The search results contain the server control for entry change notification if a nonzero value is specified for the `return_echg_controls` parameter.

Function return value

The function return value is `LDAP_SUCCESS` if no error is detected. Otherwise, it is one of the LDAP error codes listed in the `ldap.h` include file.

The following are some common client errors:

LDAP_NO_MEMORY
: Insufficient storage is available.

LDAP_PARAM_ERROR
: A parameter is not valid.

ldap_create_sort_control()

Purpose

Create a sort results request control for use with an LDAP search request

Format

```
#include <ldap.h>

int ldap_create_sort_control(
    LDAP *                  ld,
    LDAPSortKey *           sort_key_list[],
    int                     is_critical,
    LDAPControl **          control)
```

Parameters

Input

ld Specifies the LDAP handle.

sort_key_list
Specifies the *sort_key_list* created by the **ldap_create_sort_keylist()** routine. All text strings in the sort keys are in either the local EBCDIC code page or UTF-8 as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle.

is_critical
Specify `1` if this is a critical control, otherwise specify `0`.

Output

control
Returns the sort results request control. The **ldap_control_free()** routine should be called to release the control when it is no longer needed. The **ldap_insert_control()** routine can be used to add the control to a list of controls for input to the **ldap_search_ext()** or **ldap_search_ext_s()** routine.

Usage

RFC 2891: LDAP Control Extension for Server Side Sorting of Search Results provides server sorting of search results. The sort is performed based upon one or more attributes contained in the search results.

The **ldap_create_sort_control()** routine takes a *sort_key_list* as input and builds a sort results request control (`1.2.840.113556.1.4.473`). The application then adds this control to the list of controls sent to the server on an LDAP search request to request that the server sort the results. This is done using the **ldap_search_ext()** or **ldap_search_ext_s()** routine. The **ldap_create_sort_keylist()** routine can be used to create the *sort_key_list*.

The LDAP server returns a sort results response control (`1.2.840.113556.1.4.474`). The **ldap_parse_sort_control()** routine can be used to parse the control and return the sort result code and optional attribute name. The sort result code is set to `LDAP_SUCCESS` if the results were successfully sorted. Otherwise, it is set to an error code indicating the reason for the failure and, if applicable, the attribute type is set to the attribute resulting in the failure.

When chasing referrals, the client application must send an initial sort results request to each of the referral servers. The original LDAP server does not ensure

that the referral server supports the sort results request control. Multiple lists are returned to the client application, one by each referral server. It is the client application's decision as to how best to present this information to the user. The client application must turn off client referral processing to get one truly sorted list, otherwise, when chasing referrals with the sort results request control specified, the search results from multiple servers are intermixed.

The paged results control can be used with the sort results control to apply the paging capability to a set of sorted results. The sort is performed on the entire set of search results before the first page is returned to the LDAP client. See "ldap_create_page_control()" on page 52 for the description of the **ldap_create_page_control()** routine for more information about obtaining paged results. When returning paged results that have been sorted, the LDAP server returns the sort results response control with each page of sorted results.

Function return value

The function return value is `LDAP_SUCCESS` if no error is detected. Otherwise, it is one of the LDAP error codes listed in the `ldap.h` include file.

The following are some common client errors:

LDAP_NO_MEMORY
: Insufficient storage is available

LDAP_PARAM_ERROR
: A parameter is not valid

ldap_create_sort_keylist()

Purpose

Create a list of sort keys

Format

```
#include <ldap.h>

int ldap_create_sort_keylist(
    LDAPSortKey ***        sort_key_list,
    const char *           sort_string)
```

Parameters

Input

sort_string
Specifies one or more attributes to be used to sort the search results.

Output

sort_key_list
Returns the list of sort keys created from the *sort_string*. The **ldap_free_sort_keylist()** routine should be called to release the key list when it is no longer needed.

Usage

The **ldap_create_sort_keylist()** routine builds a list of LDAPSortKey structures based on the list of attributes specified in the sort string. This list can then be used as input to the **ldap_create_sort_control()** routine to create the sorted results server control.

A sort key consists of three values:

1. the name of the attribute used to sort entries returned by the server
2. the optional name of a matching rule for that attribute
3. an optional indicator of whether the sort should be done in reverse order

The sort string consists of one or more attribute specifications separated by blanks. Each attribute specification has the following format:

```
[-]attribute-type[:matching-rule]
```

The results are sorted in reverse order if the attribute specification is prefixed with a minus sign (-). The matching rule is specified by either its object identifier or its name as defined in the directory schema. The matching rule associated with the attribute type is used if no matching rule is specified.

The LDAPSortKey structure is defined as follows:

```
struct LDAPsortkey {
     char *        attr_type;
     char *        matching_rule_oid;
     int           reverse_order;
} LDAPSortKey, LDAPsortkey;
```

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes listed in the ldap.h include file.

The following are some common client errors:

LDAP_NO_MEMORY
Insufficient storage is available

LDAP_PARAM_ERROR
A parameter is not valid

ldap_delete(), ldap_delete_s(), ldap_delete_ext(), ldap_delete_ext_s()

Purpose

Delete an entry from the LDAP directory

Format

```
#include <ldap.h>

int ldap_delete(
    LDAP *                          ld,
    const char *                    dn)

int ldap_delete_s(
    LDAP *                          ld,
    const char *                    dn)

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

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

Parameters

Input

ld Specifies the LDAP handle.

dn Specifies the distinguished name for the directory entry as a null-terminated character string in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. A zero-length name is not allowed for a delete request.

serverctrls
Specifies an array of server controls for the delete request. The end of the array is indicated by a NULL address. If NULL is specified for this parameter, the server controls specified by the LDAP_OPT_SERVER_CONTROLS option for the LDAP handle are used. If NULL is specified for this parameter and the LDAP_OPT_SERVER_CONTROLS option has not been set for the LDAP handle, no server controls are used. To override the server controls for the LDAP handle so that no controls are used, specify a server controls array consisting of a NULL address. (Control values for this routine vary depending on whether you are specifying server or client controls. See "LDAP controls" on page 15 for details.)

clientctrls
Specifies an array of client controls for the delete request. The end of the array is indicated by a NULL address. If NULL is specified for this parameter, the client controls specified by the LDAP_OPT_CLIENT_CONTROLS option for the LDAP handle are used. If NULL is specified for this parameter and the LDAP_OPT_CLIENT_CONTROLS option has not been set for the LDAP handle, no client controls are used. To override the client controls for the LDAP handle so that no controls are used, specify a client controls array consisting of a NULL

address. (Control values for this routine vary depending on whether you are specifying server or client controls. See “LDAP controls” on page 15 for details.)

Output

msgidp
Returns the message identifier assigned to the delete request message. The application can use this value when calling the **ldap_result()** routine to wait for the delete result message.

Usage

The **ldap_delete()** and **ldap_delete_ext()** routines send the request to the LDAP server and return control to the application. The application must call the **ldap_result()** routine to obtain the result.

The **ldap_delete_s()** and **ldap_delete_ext_s()** routines send the request to the LDAP server and wait for the completion of the request. The delete request is abandoned if the client is unable to wait for the response because of an error from the **ldap_result()** routine.

The requested directory entry is deleted. The entry must be a leaf entry (that is, the entry must not have any subordinate entries).

Client controls specified by the LDAP_OPT_CLIENT_CONTROLS and server controls specified by the LDAP_OPT_SERVER_CONTROLS options are used by the **ldap_delete()** and **ldap_delete_s()** routines. These controls are also used by the **ldap_delete_ext()** and **ldap_delete_ext_s()** routines unless overridden by the serverctrls and clientctrls parameters.

Function return value

The **ldap_delete()** routine returns -1 if a client error is detected. Otherwise, it returns the message identifier assigned to the delete request. If the return value is -1, the application should call the **ldap_get_errno()** routine to get the error code. Errors reported by the LDAP server are not returned by the **ldap_delete()** routine. Instead, the application must call the **ldap_parse_result()** routine to obtain the result code from the result message returned by the **ldap_result()** routine.

The **ldap_delete_ext()** routine returns LDAP_SUCCESS if the request is sent to the LDAP server. Otherwise, the return value is one of the error codes listed in the ldap.h include file. Errors reported by the LDAP server are not returned by the **ldap_delete_ext()** routine. Instead, the application must call the **ldap_parse_result()** routine to obtain the result code from the result message returned by the **ldap_result()** routine.

The **ldap_delete_s()** and **ldap_delete_ext_s()** routines return LDAP_SUCCESS if the request is successful. Otherwise, the return value is one of the error codes listed in the ldap.h include file. The return value includes errors detected by the LDAP client and errors detected by the LDAP server.

The following are some common client errors:

LDAP_INVALID_STATE
An unbind request has been issued for the LDAP handle.

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_NOT_SUPPORTED
The LDAP protocol version must be LDAP_VERSION3 to specify server or client controls.

LDAP_PARAM_ERROR
A parameter is not valid.

LDAP_SERVER_DOWN
Network connection failed.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION
A critical client control is either not recognized or is not supported for a delete operation.

The following are some common server result codes:

LDAP_INSUFFICIENT_ACCESS
Not authorized to delete the directory entry.

LDAP_NO_SUCH_OBJECT
The directory entry does not exist.

LDAP_NOT_ALLOWED_ON_NONLEAF
An entry with subordinate entries cannot be deleted.

LDAP_REFERRAL
The entry is not in the current LDAP server.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION
A critical server control is either not recognized or is not supported for a delete operation.

ldap_dn2ufn()

Purpose

Parse a distinguished name and return a user-friendly name

Format

```
#include <ldap.h>

char * ldap_dn2ufn(
    const char *              dn)
```

Parameters

Input

dn Specifies the DN as a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable.

Usage

The **ldap_dn2ufn()** routine breaks a distinguished name (DN) into one or more relative distinguished names (RDN) following the rules that are defined in RFC 2253: *UTF-8 String Representation of Distinguished Names*. The RDN values are then combined into a single character string with a comma and a space between each value. Leading and trailing blanks are removed for each RDN but embedded blanks remain unchanged. Escape sequences are not removed from the attribute values.

Example: "cn=John Doe+employeeNumber=112233,ou=Manufacturing,o=Acme,c=US" is returned as "John Doe, 112233, Manufacturing, Acme, US".

Function return value

The function return value is NULL if an error is detected. Otherwise, it is the address of the user-friendly name. The application should call the **ldap_memfree()** routine to release the character string when it is no longer needed.

ldap_dn2ufn_np()

Purpose

Parse a distinguished name and return a user-friendly name

Format

```
#include <ldap.h>

int ldap_dn2ufn_np(
    LDAP *                  ld,
    const char *            dn,
    char **                 ufn)
```

Parameters

Input

ld Specifies the LDAP handle. This parameter can be specified as NULL if all text strings are in UTF-8.

dn Specifies the DN as a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle.

Output

ufn
Returns the user-friendly name as a null-terminated character string in either UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. The **ldap_memfree()** routine should be called to release the string when it is no longer needed.

Usage

The **ldap_dn2ufn()** routine breaks a distinguished name (DN) into one or more relative distinguished names (RDN) following the rules defined in RFC 2253: *UTF-8 String Representation of Distinguished Names*. The RDN values are then combined into a single character string with a comma and a space between each value. Leading and trailing blanks are removed for each RDN but embedded blanks remain unchanged. Escape sequences are not removed from the attribute values.

Example: "cn=John Doe+employeeNumber=112233,ou=Manufacturing,o=Acme,c=US" is returned as "John Doe, 112233, Manufacturing, Acme, US".

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes listed in the ldap.h include file.

The following are some common errors for this routine:

LDAP_NO_MEMORY
Insufficient storage available.

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_enetwork_domain_get()

Purpose

Return the eNetwork domain for the current user

Format

```
#include <ldap.h>

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

Parameters

Input

filename
Specifies the name of the user information file as a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable. Specify NULL for this parameter to use the default user information file ($HOME/ldap_user_info). The maximum length of the file name is 255 and a longer name is truncated to the first 255 bytes.

Output

edomainp
Returns the eNetwork domain as a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable. The application should call the **ldap_memfree()** routine to release the string when it is no longer needed.

Usage

The **ldap_enetwork_domain_get()** routine returns the eNetwork domain. The eNetwork domain can be used by the **ldap_server_locate()** routine to form the service name and allows LDAP servers within the same DNS domain to be further subdivided based on the eNetwork domain.

The eNetwork domain is set by the **ldap_enetwork_domain_set()** routine and is saved in the file that is specified by the *filename* parameter. For the default user information file, the home directory is obtained from the $HOME environment variable. If the $HOME environment variable is not defined, the home directory is obtained from the OMVS segment for the current user. The user name is indeterminate if the same UID is assigned to multiple users because the system returns the first user name with the UID you want.

Guideline: To avoid conflicts, the $HOME environment variable should always be defined for users sharing the same UID value.

The entries in the user information file are accessed by user name. This allows multiple users to share the same home directory and still have unique values for the eNetwork domain. Users with the same UID value, however, share the same entry in the user information file and have the same eNetwork domain.

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes listed in the ldap.h include file.

The following are some common errors for this routine:

LDAP_INSUFFICIENT_ACCESS
Not authorized to read the user information file.

LDAP_LOCAL_ERROR
A system routine returned an error.

LDAP_NO_MEMORY
Insufficient storage available.

LDAP_NO_SUCH_OBJECT
The user information file does not exist or the user is not defined in the user information file.

LDAP_PARAM_ERROR
A parameter is not valid.

LDAP_USER_INFO_FILE_ERROR
Error processing user information file.

ldap_enetwork_domain_set()

Purpose

Set the eNetwork domain for the current user

Format

```
#include <ldap.h>

int ldap_enetwork_domain_set(
    const char *                    edomain,
    const char *                    filename)
```

Parameters

Input

edomain
Specifies the eNetwork domain as a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable. The maximum length of the eNetwork domain is 255 and an error is returned if the name is too long. The eNetwork domain should consist of characters that can be represented in the ISO8859-1 code page to avoid problems when the eNetwork domain is later used to create a DNS resource name. Specify NULL for this parameter to indicate there is no eNetwork domain for the current user.

filename
Specifies the name of the user information file as a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable. Specify NULL for this parameter to use the default user information file ($HOME/ldap_user_info). The maximum length of the file name is 255 and a longer name is truncated to the first 255 bytes.

Usage

The **ldap_enetwork_domain_set()** routine sets the eNetwork domain. The eNetwork domain can be used by the **ldap_server_locate()** routine to form the service name and allows LDAP servers within the same DNS domain to be further subdivided based on the eNetwork domain.

The eNetwork domain is saved in the file specified by the *filename* parameter. The file is created if it does not exist. Existing domain information for the user is replaced. For the default user information file, the home directory is obtained from the $HOME environment variable. If the $HOME environment variable is not defined, the home directory is obtained from the OMVS segment for the current user. The user name is indeterminate if the same UID is assigned to multiple users because the system returns the first user name with the UID you want.

Guideline: To avoid conflicts, the $HOME environment variable should always be defined for users sharing the same UID value.

The entries in the user information file are accessed by user name. This allows multiple users to share the same home directory and still have unique values for the eNetwork domain. Users with the same UID value, however, share the same entry in the user information file and have the same eNetwork domain.

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes listed in the ldap.h include file.

The following are some common errors for this routine:

LDAP_INSUFFICIENT_ACCESS
Not authorized to read the user information file.

LDAP_LOCAL_ERROR
A system routine returned an error.

LDAP_NO_MEMORY
Insufficient storage available.

LDAP_PARAM_ERROR
A parameter is not valid.

LDAP_USER_INFO_FILE_ERROR
Error processing user information file.

ldap_err2string()

Purpose

Return a descriptive text message for an LDAP error code

Format

```
include <ldap.h>

char * ldap_err2string(
    int                 error)
```

Parameters

Input

error
Specifies the error code.

Usage

The **ldap_err2string()** routine returns a descriptive text message for an LDAP error code. The application must not modify or free this text message. The message is in the local EBCDIC code page or UTF-8, as determined by the `LDAP_LIBASCII` compiler variable

Function return value

The function return value is the address of the text message and is never a `NULL` address. The returned message is `N/A` if the LDAP message catalog cannot be accessed, storage cannot be allocated, or the error code is not a recognized LDAP error code.

ldap_explode_dn()

Purpose

Parse a distinguished name into an array of relative distinguished names

Format

```
include <ldap.h>

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

Parameters

Input

dn Specifies the distinguished name as a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable.

notypes
Specify 0 (FALSE) if the relative distinguished names should contain the attribute types and the attribute values. Specify 1 (TRUE) if the relative distinguished names should contain just the attribute values.

Usage

The **ldap_explode_dn()** routine breaks a distinguished name (DN) into one or more relative distinguished names (RDN) following the rules that are defined in RFC 2253: *UTF-8 String Representation of Distinguished Names*. Leading and trailing blanks are removed for each RDN but embedded blanks remain unchanged. Escape sequences are not removed from the attribute values. In addition, the following rules apply.

Rules

- If notypes is zero:
  - RDN values contain both the attribute types and the attributes values.
  - Opening and closing quotation marks are *not* removed from attribute values that are enclosed in quotation marks.
  - All attributes in an RDN are returned as a single array element.
- If notypes is nonzero:
  - The RDN values contain only the attribute values.
  - Opening and closing quotation marks are removed from attribute values that are enclosed in quotation marks.
  - Each attribute in an RDN is returned as a separate array element.

Example: "cn=John+sn=Doe,ou=Manufacturing,o=Acme,c=US" is parsed as follows:

- If notypes is nonzero: {"John", "Doe", "Manufacturing", "Acme", "US", NULL}
- If notypes is zero: {"cn=John+sn=Doe", "ou=Manufacturing", "o=Acme", "c=US", NULL}

Function return value

The function return value is NULL if an error is detected. Otherwise, it is the address of an array of character strings. The end of the array is indicated by a NULL address. The application should call the **ldap_value_free()** routine to release the character string array when it is no longer needed.

ldap_explode_dn_np()

Purpose

Parse a distinguished name and return an LDAP DN description

Format

```
include <ldap.h>

int ldap_explode_dn_np(
    LDAP *                      ld,
    const char *                dn,
    LDAPDNDesc **               ldnpp)
```

Parameters

Input

ld Specifies an LDAP handle. This parameter can be specified as NULL if all text strings are in UTF-8. Otherwise, all text strings are in either the local EBCDIC code page or UTF-8, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle.

dn Specifies the distinguished name as a null-terminated character string in either the local EBCDIC code page or UTF-8, as determined by the LDAP handle.

Output

ldnpp
Returns the address of the DN description. The application should call the **ldap_free_dndesc_np()** routine to release the DN description when it is no longer needed.

Usage

The **ldap_explode_dn_np()** routine breaks a distinguished name (DN) into one or more relative distinguished names (RDN) following the rules defined in RFC 2253: *UTF-8 String Representation of Distinguished Names*. Each RDN consists of one or more attributes. Leading and trailing blanks are removed from the attribute types and the attribute values (embedded blanks are not removed). In addition, the opening and closing quotation marks are removed from attribute values that are enclosed in quotation marks. Escape sequences in attribute values are processed and replaced by their equivalent UTF-8 encodings.

Example:

The following DN:

```
"cn=John+sn=Doe,ou=Manufacturing,o=Acme,c=US"
```

is parsed as shown:

```
{{{"cn", "John"}, {"sn", "Doe"}}, {{"ou", "Manufacturing"}}, {{"o", "Acme"}}, {{"c", "US"}}}
```

The LDAPDNDesc structure is defined as follows:

```
typedef struct ldap_dn_desc   {
    int                 ldn_count;
    LDAPRDNDesc *       ldn_rdns;
} LDAPDNDesc;
```

where:

ldn_count
Returns the number of RDN components in the DN.

ldn_rdns
Returns the address of an array of LDAPRDNDesc structures. There is an LDAPRDNDesc structure for each RDN component of the DN.

The LDAPRDNDesc structure is defined as follows:

```
typedef struct ldap_rdn_desc   {
    int                lrdn_count;
    LDAPAttrDesc *     lrdn_attrs;
} LDAPRDNDesc;
```

where:

lrdn_count
Returns the number of attributes in the RDN.

lrdn_attrs
Returns the address of an array of LDAPAttrDesc structures. There is an LDAPAttrDesc structure for each attribute in the RDN.

The LDAPAttrDesc structure is defined as follows:

```
typedef struct ldap_attr_desc   {
    char *             lattr_type;
    char *             lattr_value;
} LDAPAttrDesc;
```

where:

lattr_type
Returns the address of the attribute type as a null-terminated character string.

lattr_value
Returns the address of the attribute value as a null-terminated character string.

The attribute type and attribute value strings are returned in UTF-8 or the local EBCDIC code page, as determined by the LDAP handle. Escape sequences are added when converting attribute values to the local EBCDIC code page for characters that cannot be represented in the ISO8859-1 code page. (No escape sequences are needed when the strings are returned in UTF-8.)

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes listed in the ldap.h include file.

The following are some common errors for this routine:

LDAP_INVALID_DN_SYNTAX
The distinguished name is not valid.

LDAP_NO_MEMORY
Insufficient storage available.

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_explode_rdn()

Purpose

Parse a relative distinguished name into an array of attributes

Format

```
include <ldap.h>

char ** ldap_explode_rdn(
    const char *              rdn,
    int                       notypes)
```

Parameters

Input

rdn
Specifies the relative distinguished name as a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable.

notypes
Specify 0 (FALSE) if the returned attribute strings should contain the attribute types and the attribute values. Specify 1 (TRUE) if the returned attribute strings should contain just the attribute values.

Usage

The **ldap_explode_rdn()** routine breaks a relative distinguished name (RDN) into one or more attributes following the rules defined in RFC 2253: *UTF-8 String Representation of Distinguished Names*. The attribute strings contain just the attribute values if *notypes* is nonzero; otherwise the attribute strings contain both the attribute types and the attributes values. Leading and trailing blanks are removed for each attribute but embedded blanks remain unchanged. Escape sequences are not removed from the attribute values.

Example: "cn=John+sn=Doe " is parsed as follows:

- If *notypes* is nonzero: {"John", "Doe", NULL}
- If *notypes* is zero: {"cn=John", "sn=Doe", NULL}

Function return value

The function return value is NULL if an error is detected. Otherwise, it is the address of an array of character strings. The end of the array is indicated by a NULL address. The application should call the **ldap_value_free()** routine to release the character string array when it is no longer needed.

ldap_extended_operation(), ldap_extended_operation_s()

Purpose

Perform extended operations

Format

```
#include <ldap.h>

int ldap_extended_operation(
    LDAP *                          ld,
    const char *                    reqoid,
    const BerVal *                  reqdata,
    LDAPControl *                   serverctrls[],
    LDAPControl *                   clientctrls[],
    int *                           msgidp)

int ldap_extended_operation_s(
    LDAP *                          ld,
    const char *                    reqoid,
    const BerVal *                  reqdata,
    LDAPControl *                   serverctrls[],
    LDAPControl *                   clientctrls[],
    char **                         resultoidp,
    BerVal **                       resultdatap)
```

Parameters

Input

ld Specifies the LDAP handle.

reqoid
Specifies the extended operation request OID as a dotted decimal null-terminated character string in either UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle.

reqdata
Specifies the data for the extended operation request. Specify NULL for this parameter if no data is needed for the request. The data must be in the correct format for the extended operation request and is not modified by the LDAP client.

serverctrls
Specifies an array of server controls for the extended operation request. The end of the array is indicated by a NULL address. If NULL is specified for this parameter, the server controls specified by the LDAP_OPT_SERVER_CONTROLS option for the LDAP handle are used. If NULL is specified for this parameter and the LDAP_OPT_SERVER_CONTROLS option has not been set for the LDAP handle, no server controls are used. To override the server controls for the LDAP handle so that no controls are used, specify a server controls array consisting of a NULL address. (Control values for this routine vary depending on whether you are specifying server or client controls. See “LDAP controls” on page 15 for details.)

clientctrls
Specifies an array of client controls for the extended operation request. The end of the array is indicated by a NULL address. If NULL is specified for this parameter, the client controls specified by the LDAP_OPT_CLIENT_CONTROLS option for the LDAP handle are used. If NULL is specified for this parameter and the LDAP_OPT_CLIENT_CONTROLS option has not been set for the LDAP

handle, no client controls are used. To override the client controls for the LDAP handle so that no controls are used, specify a client controls array consisting of a NULL address. (Control values for this routine vary depending on whether you are specifying server or client controls. See "LDAP controls" on page 15 for details.)

Output

msgidp
Returns the message identifier assigned to the extended operation request message. The application can use this value when calling the **ldap_result()** routine to wait for the extended operation result message.

resultoidp
Returns the extended operation result OID as a dotted decimal null-terminated character string in either UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. The application should call the **ldap_memfree()** routine to release the OID string when it is no longer needed. The returned value is NULL if the extended operation result did not contain an OID. Specify NULL for this parameter if the OID should not be returned.

resultdatap
Returns the data from the extended operation result. The application should call the **ldap_berfree_np()** routine to release the result data when it is no longer needed. The returned value is NULL if the extended operation result did not contain any data. Specify NULL for this parameter if the result data should not be returned.

Usage

The **ldap_extended_operation()** and **ldap_extended_operation_s()** routines perform an extended operation targeted at the LDAP server. The LDAP protocol version must be LDAP_VERSION3 in order to perform an extended operation. The extended operations that are available depend upon the LDAP server. The **supportedExtension** attribute in the root DSE can be queried to determine if the LDAP server supports a particular extended operation.

The **ldap_extended_operation()** routine initiates the extended operation and returns control to the application. The application must call the **ldap_result()** routine to wait for the completion of the extended operation. The application can call the **ldap_parse_extended_result()** routine to obtain the result OID and any result data from the result message returned by the **ldap_result()** routine.

The **ldap_extended_operation_s()** routine initiates the extended operation and waits for it to complete. The extended operation request is abandoned if the client is unable to wait for the response because of an error from the **ldap_result()** routine.

Function return value

The function return value is LDAP_SUCCESS if the request is successful. Otherwise, it is one of the errors listed in the ldap.h include file. The **ldap_extended_operation()** routine returns only errors detected by the client run time. The **ldap_extended_operation_s()** routine returns errors detected by both the client run time and the LDAP server.

The following are some common client errors:

LDAP_INVALID_STATE
An unbind request has been issued for the LDAP handle.

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_NOT_SUPPORTED
The LDAP protocol version must be LDAP_VERSION3 to initiate an extended operation.

LDAP_PARAM_ERROR
A parameter is not valid.

LDAP_SERVER_DOWN
Network connection failed.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION
A critical client control is either not recognized or is not supported for an extended operation.

The following are some common server result codes:

LDAP_PROTOCOL_ERROR
The server does not support the requested extended operation.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION
A critical server control is either not recognized or is not supported for an extended operation.

ldap_first_attribute()

Purpose

Return the attribute type for the first attribute in an LDAP search entry

Format

```
#include <ldap.h>

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

Parameters

Input

ld Specifies the LDAP handle.

entry
Specifies an entry returned by the **ldap_first_entry()** or **ldap_next_entry()** routine.

Output

ber
Returns the address of an LDAP control block used to maintain the current attribute position. The application must not modify this control block.

Usage

The **ldap_first_attribute()** routine returns the attribute type for the first attribute in the search entry. The **ldap_next_attribute()** routine should be called to obtain successive attributes in the search entry. The **ldap_get_values()** or **ldap_get_values_len()** routine can be called to get the attribute values associated with the attribute type.

The *ber* parameter returns the address of a control block allocated and maintained by the LDAP client run time. This control block is released when the **ldap_next_attribute()** routine returns a NULL value. The application should call the **ldap_memfree()** routine to release this control block if the application does not want to keep calling the **ldap_next_attribute()** routine until all attributes have been processed.

Function return value

The function return value is the attribute type of the first attribute. The attribute type is a null-terminated character string in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. The application should call the **ldap_memfree()** routine to release the attribute type when it is no longer needed. The return value is NULL if there are no attributes or if an error is detected. The **ldap_get_errno()** routine can be called to get the error code when the return value is NULL. The error code is LDAP_SUCCESS if there are no attributes.

The following are some common errors for this routine:

LDAP_NO_MEMORY
Insufficient storage available.

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_first_entry()

Purpose

Return the first search entry in an LDAP result

Format

```
#include <ldap.h>

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

Parameters

Input

ld Specifies the LDAP handle.

result
Specifies the result message returned by **ldap_result()** or one of the synchronous request routines.

Usage

The **ldap_first_entry()** routine returns the address of the first search entry in the LDAP result. The **ldap_next_entry()** routine should be called to obtain successive entries in the LDAP result.

Function return value

The function return value is the address of the first search entry. The return value is NULL if there are no search entries or if an error is detected. The **ldap_get_errno()** routine can be called to get the error code when the return value is NULL. The error code is LDAP_SUCCESS if there are no search entries.

The following is a common error for this routine:

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_first_message()

Purpose

Return the first message in an LDAP result

Format

```
#include <ldap.h>

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

Parameters

Input

ld Specifies the LDAP handle.

result
Specifies the result message returned by **ldap_result()** or one of the synchronous request routines.

Usage

The **ldap_first_message()** routine returns the address of the first message in the LDAP result. Call the **ldap_next_message()** routine to obtain successive messages in the LDAP result.

Function return value

The function return value is the address of the first message. The return value is NULL if an error is detected. The **ldap_get_errno()** routine can be called to get the error code when the return value is NULL.

The following is a common error for this routine:

LDAP_PARAM_ERROR
A parameter is not valid; for example, there is no message in the result.

ldap_first_reference()

Purpose

Return the first search reference in an LDAP result

Format

```
#include <ldap.h>

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

Parameters

Input

ld Specifies the LDAP handle.

result
Specifies the result message returned by **ldap_result()** or one of the synchronous request routines.

Usage

The **ldap_first_reference()** routine returns the address of the first search reference in the LDAP result. The **ldap_next_reference()** routine should be called to obtain successive references in the LDAP result.

Function return value

The function return value is the address of the first search reference. The return value is NULL if there are no search references or if an error is detected. The **ldap_get_errno()** routine can be called to get the error code when the return value is NULL. The error code is LDAP_SUCCESS if there are no search references.

The following is a common error for this routine:

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_free_dndesc_np()

Purpose

Release storage allocated for an LDAP DN description

Format

```
#include <ldap.h>

void ldap_free_dndesc_np(
    LDAPDNDesc *          ldnp)
```

Parameters

Input

ldnp
Specifies the LDAP DN description to be released.

Usage

The **ldap_free_dndesc_np()** routine releases the storage allocated for an LDAP DN (Distinguished Name) description returned by the **ldap_explode_dn_np()** routine.

Function return value

There is no function return value.

ldap_free_sort_keylist()

Purpose

Release storage allocated for a list of sort keys

Format

```
#include <ldap.h>

void ldap_free_sort_keylist(
    LDAPSortKey *          sort_key_list[])
```

Parameters

Input

sort_key_list
Specifies the sort key list to be released.

Usage

The **ldap_free_sort_keylist()** routine releases the storage allocated for a list of sort keys created by the **ldap_create_sort_keylist()** routine. The address array and each LDAPSortKey structure are released along with the associated data objects.

Function return value

There is no function return value.

ldap_free_urldesc()

Purpose

Release storage allocated for an LDAP URL description

Format

```
#include <ldap.h>

void ldap_free_urldesc(
    LDAPURLDesc *          ludp)
```

Parameters

Input

ludp
Specifies the LDAP URL description to be released.

Usage

The **ldap_free_urldesc()** routine releases the storage allocated for an LDAP URL description returned by the **ldap_url_parse()** or **ldap_url_parse_np()** routine.

Function return value

There is no function return value.

ldap_get_dn()

Purpose

Return the distinguished name from the search entry

Format

```
#include <ldap.h>

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

Parameters

Input

ld Specifies the LDAP handle.

entry
Specifies a search entry returned by the **ldap_first_entry()** or **ldap_next_entry()** routine.

Usage

The **ldap_get_dn()** routine returns the distinguished name from a search entry.

Function return value

The function return value is the address of the distinguished name for the entry or NULL if an error is detected. The name is returned as a null-terminated character string in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. The application should call the **ldap_memfree()** routine to release the name when it is no longer needed. The **ldap_get_errno()** routine can be called to obtain the error code when the return value is NULL.

The following are some common errors for this routine:

LDAP_NO_MEMORY
Insufficient storage available.

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_get_entry_controls_np()

Purpose

Return the server controls from a search entry message

Format

```
#include <ldap.h>

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

Parameters

Input

ld Specifies the LDAP handle.

entry
Specifies an entry returned by the **ldap_first_entry()** or **ldap_next_entry()** routine.

Output

serverctrlsp
Returns the server controls as an array of `LDAPControl` structures. The end of the array is indicated by a `NULL` control address. The return value is `NULL` if the LDAP server did not return any server controls. The control OID string is in UTF-8 or the local EBCDIC code page, as determined by the `LDAP_OPT_UTF8_IO` option for the LDAP handle. The control value is unchanged and has the format returned by the LDAP server. The application should call the **ldap_controls_free()** routine to release the controls array when it is no longer needed. (Control values for this routine vary depending on whether you are specifying server or client controls. See “LDAP controls” on page 15 for details.)

Usage

The **ldap_get_entry_controls_np()** routine returns the server controls from a search entry message. A parameter error is returned if the message is not a search entry message.

Function return value

The function return value is `LDAP_SUCCESS` if no error is detected. Otherwise, it is one of the LDAP error codes listed in the `ldap.h` include file.

The following are some common errors for this routine:

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_get_errno()

Purpose

Return the last error code for an LDAP handle

Format

```
#include <ldap.h>

int ldap_get_errno(
    LDAP *               ld)
```

Parameters

Input

ld Specifies the LDAP handle.

Usage

The **ldap_get_errno()** routine returns the last error associated with an LDAP handle. In a multi-threaded environment, this is the error for the last request using the LDAP handle and not necessarily the last request issued by the current thread. The error code associated with the LDAP handle is not reset by a successful LDAP request and remains unchanged until the next error is detected.

Function return value

The function return value is `LDAP_SUCCESS` if no error has been detected. Otherwise, it is one of the LDAP error codes listed in the `ldap.h` include file.

ldap_get_function_vector()

Purpose

Obtain the address of the LDAP function vector

Format

```
#include <ldap.h>

void ldap_get_function_vector(
    unsigned int *        function_mask,
    LDAPFunctions **      function_vector)
```

Parameters

Output

function_mask
Returns a bit mask indicating the LDAP API level.

function_vector
Returns the address of the LDAP function vector. The LDAP function vector for native ASCII mode is returned if the LDAP_LIBASCII compiler variable is defined. Otherwise, the LDAP function vector for native EBCDIC mode is returned.

Usage

LDAP functions can be called using either static binding or runtime binding. Static binding is performed when the application is compiled, while runtime binding is performed when the application is run.

In order to use static binding, the LDAP sidefile is specified as input to the binder. This causes all LDAP functions to be resolved at bind time and causes the LDAP client DLL to be implicitly loaded when the application is run.

In order to use runtime binding, the LDAP client DLL must be explicitly loaded by the application and the LDAP functions must be called using indirect addresses. The **ldap_get_function_vector()** routine allows an application to obtain the address of the LDAP function vector containing an entry for each LDAP API routine. This eliminates the need for the application to build the function vector through repeated calls to the **dllqueryfn()** routine.

The function mask indicates the capabilities of the LDAP client DLL. The following values have been defined:

LDAP_API_LVL1
LDAP functions provided as part of z/OS Version 1 Release 6 and 7 are available.

LDAP_API_LVL2
LDAP functions provided as part of z/OS Version 1 Release 8, 9, and 10 are available.

LDAP_API_LVL3
LDAP functions provided as part of z/OS Version 1 Release 11 are available.

LDAP_API_LVL4
LDAP functions provided as part of z/OS Version 1 Release 12, z/OS Version 1 Release 13, z/OS Version 2 Release 1, or z/OS Version 2 Release 2, or higher are available.

Function return value

There is no function return value.

ldap_get_lderrno()

Purpose

Return information for the most recent error

Format

```
#include <ldap.h>

int ldap_get_lderrno(
    LDAP *              ld,
    char **             matcheddnp,
    char **             errmsgp)
```

Parameters

Input

ld Specifies the LDAP handle.

Output

matcheddnp
Returns the matched distinguished name from the most recent result message as a null-terminated character string. The string is in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. The return value is NULL if the most recent result message does not contain a matched distinguished name. The application should call the **ldap_memfree()** routine to release the string when it is no longer needed. Specify NULL for this parameter if the matched distinguished name should not be returned.

errmsgp
Returns the error text from the most recent result message as a null-terminated character string. The string is in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. The return value is NULL if the LDAP server did not return any error text. The application should call the **ldap_memfree()** routine to release the string when it is no longer needed. Specify NULL for this parameter if the error text should not be returned.

Usage

The **ldap_get_lderrno()** routine obtains information for the most recent error that occurred for an LDAP operation. In a multi-threaded environment, this might not be an error from a request issued by this thread. When an error occurs on the LDAP server, the server returns the following information to the client:

- The LDAP result code for the error that occurred.
- A message containing any additional information about the error from the server.
- A matched distinguished name (DN), which identifies a portion of an existing entry, might be returned if the DN specified on the last LDAP operation does not exist on the LDAP server.

Function return value

The function return value is the LDAP result code from the most recent error.

ldap_get_option()

Purpose

Return the value for an LDAP option

Format

```
#include <ldap.h>

int ldap_get_option(
    LDAP *              ld,
    int                 option,
    void *              value)
```

Parameters

Input

ld Specifies the LDAP handle.

option
Specifies the option identifier.

Output

value
Returns the option value.

Usage

The **ldap_get_option()** routine returns the value of an LDAP option for the supplied LDAP handle. The manner in which the option value is returned depends upon the option type. Table 2 summarizes how the options are returned.

Table 2. How ldap_get_option values are returned

Option	Value parameter
LDAP_OPT_CLIENT_CONTROLS	LDAPControl ***
LDAP_OPT_CONNECT	int *
LDAP_OPT_DEBUG	int *
LDAP_OPT_DEBUG_FILENAME	char **
LDAP_OPT_DEBUG_STRING	char **
LDAP_OPT_DELEGATION	int *
LDAP_OPT_DEREF	int *
LDAP_OPT_ERROR_NUMBER	int *
LDAP_OPT_ERROR_STRING	char **
LDAP_OPT_EXT_ERROR	int *
LDAP_OPT_EXT_REBIND_FN	LDAPExtRebindProc *
LDAP_OPT_HOST_NAME	char **
LDAP_OPT_IO_CALLBACK	LDAPIOCallback *
LDAP_OPT_MATCHED_DN	char **
LDAP_OPT_MAX_SASL_LEVEL	int *
LDAP_OPT_MIN_SASL_LEVEL	int *

Table 2. How ldap_get_option values are returned (continued)

Option	Value parameter
LDAP_OPT_PROTOCOL_VERSION	int *
LDAP_OPT_REBIND_FN	LDAPRebindProc *
LDAP_OPT_REFERRALS	int *
LDAP_OPT_REFHOPLIMIT	int *
LDAP_OPT_RESTART	int *
LDAP_OPT_SASL_QOP	int *
LDAP_OPT_SERVER_CONTROLS	LDAPControl ***
LDAP_OPT_SIZELIMIT	int *
LDAP_OPT_SOCKS_CONF	char **
LDAP_OPT_SOCKS_PASSWORD	char **
LDAP_OPT_SOCKS_SERVER	char **
LDAP_OPT_SOCKS_USERNAME	char **
LDAP_OPT_SOCKS_VERSION	int *
LDAP_OPT_SSL	int *
LDAP_OPT_SSL_CIPHER	char **
LDAP_OPT_SSL_CIPHER_EXPANDED	char **
LDAP_OPT_SSL_CIPHER_FORMAT	int *
LDAP_OPT_SSL_TIMEOUT	int *
LDAP_OPT_TIMELIMIT	int *
LDAP_OPT_UTF8_IO	int *
LDAP_OPT_V2_WIRE_FORMAT	int *

For example, the LDAP_OPT_SIZELIMIT option is returned as follows:

```
int sizeLimit;
ldap_get_option(ld, LDAP_OPT_SIZELIMIT, &sizeLimit);
```

The following LDAP options can be returned:

- **LDAP_OPT_CLIENT_CONTROLS**

  The LDAP_OPT_CLIENT_CONTROLS option returns the address of a list of client controls to be processed with each request. The end of the list is indicated by a NULL control address. The list address is NULL if there are no client controls. The **ldap_controls_free()** routine should be called to release the controls when they are no longer needed. A parameter error is returned if the LDAP protocol version is not set to LDAP_VERSION3.

  The OID string in the client control is a null-terminated character string in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. In addition, a client control value that is a character string is also in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle.

- **LDAP_OPT_CONNECT**

  The LDAP_OPT_CONNECT option returns LDAP_OPT_ON if a connection has been established with the LDAP server and LDAP_OPT_OFF otherwise. The LDAP_OPT_SSL option can be used to determine if the connection is using SSL.

- **LDAP_OPT_DEBUG**

The LDAP_OPT_DEBUG option returns a bitmap indicating the debug trace level for the LDAP client run time. The debug trace level applies to the entire process and not just the LDAP handle. For this reason, the LDAP handle can be specified as NULL. If specified, the LDAP handle must be a valid handle. If tracing is not active, the debug trace level is LDAP_DEBUG_OFF.

The debug trace level is formed by **OR**ing together one or more of the following debug options:

LDAP_DEBUG_ACL
Trace ACL processing

LDAP_DEBUG_ALL
Enable all debug traces (same as LDAP_DEBUG_ANY)

LDAP_DEBUG_ANY
Enable all debug traces (same as LDAP_DEBUG_ALL)

LDAP_DEBUG_ARGS
Trace request arguments

LDAP_DEBUG_BE_CAPABILITIES
Trace backend capabilities

LDAP_DEBUG_BER
Trace ASN.1 encode and decode processing

LDAP_DEBUG_CACHE
Trace cache activity

LDAP_DEBUG_CONNS
Trace connection activity

LDAP_DEBUG_ERROR
Trace errors

LDAP_DEBUG_FILTER
Trace filter processing

LDAP_DEBUG_INFO
Trace informational messages

LDAP_DEBUG_LDAPBE
Trace server backend activity

LDAP_DEBUG_LDBM
Trace file backend activity

LDAP_DEBUG_MESSAGE
Trace message processing

LDAP_DEBUG_MULTISERVER
Trace multiple server activity

LDAP_DEBUG_OFF
Disable all debug traces

LDAP_DEBUG_PACKETS
Trace packet activity

LDAP_DEBUG_PARSE
Trace parsing activity

LDAP_DEBUG_PERFORMANCE
Trace performance statistics

LDAP_DEBUG_PLUGIN
Trace plug-in extension activity

LDAP_DEBUG_REFERRAL
Trace referral activity

LDAP_DEBUG_REPLICATION
Trace replication activity

LDAP_DEBUG_SCHEMA
Trace schema processing

LDAP_DEBUG_SDBM
Trace RACF backend activity

LDAP_DEBUG_STATS
Trace operational statistics

LDAP_DEBUG_STRBUF
Trace and UTF-8 activity

LDAP_DEBUG_SYSPLEX
Trace sysplex activity

LDAP_DEBUG_TDBM
Trace TDBM database processing

LDAP_DEBUG_THREAD
Trace thread activity

LDAP_DEBUG_TRACE
Trace API routine entry and exit

For more information about the LDAP trace options, see "Enabling tracing" on page 242.

- **LDAP_OPT_DEBUG_FILENAME**

  The LDAP_OPT_DEBUG_FILENAME option returns the name of the LDAP trace output file. The return value is NULL if the debug file name has not been set. The application should call the **ldap_memfree()** routine to release the file name when it is no longer needed. The debug file name applies to the entire process and not just the LDAP handle. For this reason, the LDAP handle can be specified as NULL. If specified, the LDAP handle must be a valid handle. The file name is in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable.

- **LDAP_OPT_DEBUG_STRING**

  The LDAP_OPT_DEBUG_STRING option returns the active LDAP trace options as a null-terminated character string. The **ldap_memfree()** routine should be called to release the options string when it is no longer needed. The debug trace level applies to the entire process and not just the LDAP handle. For this reason, the LDAP handle can be specified as NULL. If specified, the LDAP handle must be a valid handle. The options string is in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable.

- **LDAP_OPT_DELEGATION**

  The LDAP_OPT_DELEGATION option returns LDAP_OPT_ON if the LDAP client passes Kerberos delegated credentials to the LDAP server, and LDAP_OPT_OFF otherwise. A parameter error is returned if the LDAP protocol version is not set to LDAP_VERSION3.

- **LDAP_OPT_DEREF**

  The LDAP_OPT_DEREF option returns how the LDAP server handles aliases during search requests and is one of the following values:

LDAP_DEREF_NEVER
Do not dereference aliases. (This is the default.)

LDAP_DEREF_SEARCHING
Dereference aliases in subordinates of the base object in searching but not in locating the base object of the search.

LDAP_DEREF_FINDING
Dereference aliases in locating the base object of the search but not when searching subordinates of the base object.

LDAP_DEREF_ALWAYS
Dereference aliases both in searching and in locating the base object of the search.

- **LDAP_OPT_ERROR_NUMBER**

  The LDAP_OPT_ERROR_NUMBER option returns the last error that is associated with the LDAP handle. In a multi-threaded environment, this is the error for the last request using the LDAP handle and not necessarily the last request that is issued by the current thread. The error code that is associated with the LDAP handle is not reset by a successful LDAP request and remains unchanged until the next error is detected. The value that is returned by the LDAP_OPT_ERROR_NUMBER option is the same as the value returned by the **ldap_get_errno()** routine.

- **LDAP_OPT_ERROR_STRING**

  The LDAP_OPT_ERROR_STRING option returns the error message from the most recent result message that is processed by the **ldap_result2error()** routine or by one of the synchronous request routines. In a multi-threaded environment, this might not be a result message from a request that is issued by this thread. The return value is NULL if there is no error message. The error message that is associated with the LDAP handle is reset by a successful synchronous request routine, by **ldap_result2error()** before it processes the result message, and by a routine processing an operation (such as search or modify) when a client error occurs. The returned text string is in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable. The **ldap_memfree()** routine should be called to release the error message when it is no longer needed.

- **LDAP_OPT_EXT_ERROR**

  The LDAP_OPT_EXT_ERROR option returns the last extended error code that is associated with an LDAP handle. In a multi-threaded environment, this is the error for the last request using the LDAP handle and not necessarily the last request that is issued by the current thread. The extended error code is set each time that an extended error occurs for an LDAP handle and is not reset by a successful LDAP request; it remains unchanged until the next error is detected for the LDAP handle. If there is no extended error code that is associated with the LDAP error, the extended error code is set to 0.

- **LDAP_OPT_EXT_REBIND_FN**

  The LDAP_OPT_EXT_REBIND_FN option returns the address of the routine to be called by the LDAP client run time when it must authenticate a connection with another LDAP server. (For more information about the rebind routine, see “Rebinding while following referrals” on page 12.) The return value is NULL if the LDAP_OPT_EXT_REBIND_FN option has not been set.

- **LDAP_OPT_HOST_NAME**

  The LDAP_OPT_HOST_NAME option returns the host name list for the LDAP handle. This is a null-terminated character string consisting of one or more *host:port* values separated by blanks. The application should call the **ldap_memfree()**

routine to release the string when it is no longer needed. The host name list is in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable.

- **LDAP_OPT_IO_CALLBACK**

  The LDAP_OPT_IO_CALLBACK option returns the current callback routines for the LDAP handle. For more information about the callback routines, see the description of LDAP_OPT_IO_CALLBACK for the **ldap_set_option()** and **ldap_set_option_np()** routines in "ldap_set_option(), ldap_set_option_np()" on page 182.

- **LDAP_OPT_MATCHED_DN**

  The LDAP_OPT_MATCHED_DN option returns the matched DN from the most recent result message that is processed by the **ldap_result2error()** routine or by one of the synchronous request routines. In a multi-threaded environment, this might not be a result message from a request that is issued by this thread. The return value is NULL if there is no matched DN. The matched DN associated with the LDAP handle is reset by a successful synchronous request routine, by **ldap_result2error()** before it processes the result message, and by a routine processing an operation (such as search or modify) when a client error occurs. The returned text string is in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable. The **ldap_memfree()** routine should be called to release the matched DN value when it is no longer needed.

- **LDAP_OPT_MAX_SASL_LEVEL**

  The LDAP_OPT_MAX_SASL_LEVEL option returns the maximum SASL protection level for the LDAP handle. This level is the highest SASL protection level that can be negotiated during a bind using a SASL mechanism. The negotiated protection level cannot be greater than this level even if the server offers a higher protection level. LDAP_PARAM_ERROR is returned if the LDAP protocol version is not LDAP_VERSION3.

  The SASL protection levels, in increasing level of protection, are:

  LDAP_SASL_LEVEL_NONE
  : No integrity or confidentiality protection.

  LDAP_SASL_LEVEL_INTEG
  : Integrity protection.

  LDAP_SASL_LEVEL_CONF
  : Integrity and confidentiality protection. (This is the default.)

- **LDAP_OPT_MIN_SASL_LEVEL**

  The LDAP_OPT_MIN_SASL_LEVEL option returns the minimum SASL protection level for the LDAP handle. This level is the lowest SASL protection level that can be negotiated during a bind using a SASL mechanism. The bind fails if the server does not offer at least this protection level. LDAP_PARAM_ERROR is returned if the LDAP protocol version is not LDAP_VERSION3.

  The SASL protection levels, in increasing level of protection, are:

  LDAP_SASL_LEVEL_NONE
  : No integrity or confidentiality protection. (This is the default.)

  LDAP_SASL_LEVEL_INTEG
  : Integrity protection.

  LDAP_SASL_LEVEL_CONF
  : Integrity and confidentiality protection.

- **LDAP_OPT_PROTOCOL_VERSION**

The LDAP_OPT_PROTOCOL_VERSION option returns the LDAP protocol version that is used by the LDAP client when connecting to an LDAP server and is either LDAP_VERSION2 or LDAP_VERSION3.

- **LDAP_OPT_REBIND_FN**

  The LDAP_OPT_REBIND_FN option returns the address of the routine to be called by the LDAP client run time when it must authenticate a connection with another LDAP server. (For more information about the rebind routine, see "Rebinding while following referrals" on page 12.) The return value is NULL if the LDAP_OPT_REBIND_FN option has not been set and **ldap_set_rebind_proc()** has not been called.

- **LDAP_OPT_REFERRALS**

  The LDAP_OPT_REFERRALS option returns LDAP_OPT_ON if the LDAP client follows referrals that are returned by the LDAP server and LDAP_OPT_OFF otherwise.

- **LDAP_OPT_REFHOPLIMIT**

  The LDAP_OPT_REFHOPLIMIT option returns the maximum number of LDAP servers to contact when following a referral. For subtree searches, this is the limit on the depth of nested search references, so the number of servers that are contacted might actually exceed this value.

- **LDAP_OPT_RESTART**

  The LDAP_OPT_RESTART option returns LDAP_OPT_ON if the **select()** system call should be restarted when it is interrupted by the system and LDAP_OPT_OFF otherwise.

- **LDAP_OPT_SASL_QOP**

  The LDAP_OPT_SASL_QOP option returns the quality-of-protection (QOP) negotiated between the LDAP client and the LDAP server. The QOP consists of two 16-bit fields: The upper 16 bits contain the confidentiality level and the lower 16 bits contain the integrity level. The LDAP_SASL_INTEG_MASK and LDAP_SASL_CONF_MASK masks can be used to isolate the integrity and confidentiality levels for comparison purposes. The integrity service ensures that messages are not modified or lost. The confidentiality service encrypts messages so they can be read only by the remote partner.

  The following integrity levels are supported:

  LDAP_SASL_INTEG_NONE
  No integrity service is available.

  LDAP_SASL_INTEG_MD5
  Integrity checking provided using MD5 digests.

  LDAP_SASL_INTEG_SHA1
  Integrity checking provided using SHA-1 digests.

  The following confidentiality levels are supported:

  LDAP_SASL_CONF_NONE
  No confidentiality service is available.

  LDAP_SASL_CONF_RC4_128
  Confidentiality using 128-bit RC4.

  LDAP_SASL_CONF_DES_56
  Confidentiality using 56-bit DES.

  LDAP_SASL_CONF_3DES_112
  Confidentiality using 112-bit 3DES.

  LDAP_SASL_CONF_3DES-168
  Confidentiality using 168-bit 3DES.

- **LDAP_OPT_SERVER_CONTROLS**

  The LDAP_OPT_SERVER_CONTROLS option returns the address of a default list of server controls to be sent with each request. The end of the list is indicated by a NULL control address. The return value is NULL if there are no default server controls. A parameter error is returned if the LDAP protocol version is not set to LDAP_VERSION3. The **ldap_controls_free()** routine should be called to release the controls when they are no longer needed.

  The OID string in the server control is a null-terminated character string in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. The value in the server control is returned unchanged.

- **LDAP_OPT_SIZELIMIT**

  The LDAP_OPT_SIZELIMIT option specifies the maximum number of entries that can be returned for a search request. The LDAP server can also provide a size limit on the number of entries returned. For information about the server's size limit and how it interacts with the client size limit, see the documentation for your LDAP server. For the IBM Tivoli Directory Server for z/OS, see the description of the **sizeLimit** configuration file option (Customizing the LDAP server configuration) in *z/OS IBM Tivoli Directory Server Administration and Use for z/OS*. The default size limit for the client, which is specified by a value of 0, indicates that the maximum number of entries is limited only by the LDAP server limit.

- **LDAP_OPT_SOCKS_CONF**

  The LDAP_OPT_SOCKS_CONF option returns the name of the SOCKS configuration file as a null-terminated string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable. The return value is NULL if the LDAP_OPT_SOCKS_CONF option has not been set and the SOCKS_CONF environment variable was not defined when the LDAP handle was initialized. The **ldap_memfree()** routine should be called to release the string when it is no longer needed. Note setting the LDAP_OPT_SOCKS_SERVER option clears the LDAP_OPT_SOCKS_CONF option.

- **LDAP_OPT_SOCKS_PASSWORD**

  The LDAP_OPT_SOCKS_PASSWORD option returns the SOCKS password as a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable. The return value is NULL if the LDAP_OPT_SOCKS_PASSWORD option has not been set and the SOCKS_PASSWORD environment variable was not defined when the LDAP handle was initialized. The **ldap_memfree()** routine should be called to release the string when it is no longer needed.

- **LDAP_OPT_SOCKS_SERVER**

  The LDAP_OPT_SOCKS_SERVER option returns the SOCKS server list as a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable. Entries in the character string are separated by commas. The return value is NULL if the LDAP_OPT_SOCKS_SERVER option has not been set and the SOCKS_SERVER environment variable was either not defined or was overridden by the SOCKS_CONF environment variable when the LDAP handle was initialized. The **ldap_memfree()** routine should be called to release the string when it is no longer needed. Note setting the LDAP_OPT_SOCKS_CONF option clears the LDAP_OPT_SOCKS_SERVER option.

- **LDAP_OPT_SOCKS_USERNAME**

  The LDAP_OPT_SOCKS_USERNAME option returns the SOCKS user name as a null-terminated character string in the local EBCDIC code page or UTF-8, as

determined by the LDAP_LIBASCII compiler variable. The return value is NULL if the LDAP_OPT_SOCKS_USERNAME option has not been set and the SOCKS_USERNAME environment variable was not defined when the LDAP handle was initialized. The **ldap_memfree()** routine should be called to release the string when it is no longer needed.

- **LDAP_OPT_SOCKS_VERSION**

  The LDAP_OPT_SOCKS_VERSION option returns the SOCKS protocol version, and is 4 or 5. Note the SOCKS version 5 protocol is always used when the LDAP server address is an IPv6 address, even though the LDAP_OPT_SOCKS_VERSION option is set to 4.

- **LDAP_OPT_SSL**

  The LDAP_OPT_SSL option returns LDAP_OPT_ON if an SSL connection can be used to bind to the LDAP server and LDAP_OPT_OFF otherwise. The LDAP_OPT_CONNECT option can be used to determine if a connection has been established with the LDAP server.

- **LDAP_OPT_SSL_CIPHER**

  This option is pertinent provided 2-byte SSL ciphers are currently in effect, which is based on the setting of LDAP_OPT_SSL_CIPHER_FORMAT.

  The LDAP_OPT_SSL_CIPHER option returns a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable. The string consists of a single cipher specification if an SSL connection is established with the LDAP server and provided 2-byte SSL ciphers are in effect. Otherwise, the string consists of one or more cipher suites to be used when negotiating an SSL connection with the LDAP server. The return value is NULL if an SSL connection is not opened and no cipher suites are set by the application. The returned character string consists of the cipher suites that are specified as two hexadecimal digits per cipher suite. Cipher suite values are concatenated, with no separators. The application should call the **ldap_memfree()** routine to release the string when it is no longer needed.

  Table 3 lists the cipher suites that are defined for coding convenience in the ldap.h include file. Only 2-byte cipher suites that are supported in SSL V3 and TLS V1.0 are provided in ldap.h and Table 3. For newer cipher suites supported in later TLS protocols, see *z/OS Cryptographic Services System SSL Programming*.

Table 3. SSL V3 and TLS V1.0 cipher suites

Mnemonic	Value	Description
LDAP_SSL_RC4_MD5_EX	"03"	40-bit RC4 encryption with MD5 digest and RSA key exchange
LDAP_SSL_RC4_MD5_US	"04"	128-bit RC4 encryption with MD5 digest and RSA key exchange
LDAP_SSL_RC4_SHA_US	"05"	128-bit RC4 encryption with SHA-1 digest and RSA key exchange
LDAP_SSL_RC2_MD5_EX	"06"	40-bit RC2 encryption with MD5 digest and RSA key exchange
LDAP_SSL_DES_SHA_EX	"09"	56-bit DES encryption with SHA-1 digest and RSA key exchange
LDAP_SSL_3DES_SHA_US	"0A"	168-bit 3DES encryption with SHA-1 digest and RSA key exchange
LDAP_SSL_DH_DES_SHA_DSS_EX	"0C"	56-bit DES encryption with SHA-1 digest and fixed Diffie-Hellman key exchange using DSS certificate

Table 3. SSL V3 and TLS V1.0 cipher suites (continued)

Mnemonic	Value	Description
LDAP_SSL_DH_3DES_SHA_DSS	"0D"	168-bit 3DES encryption with SHA-1 digest and fixed Diffie-Hellman key exchange using DSS certificate
LDAP_SSL_DH_DES_SHA_RSA_EX	"0F"	56-bit DES encryption with SHA-1 digest and fixed Diffie-Hellman key exchange using RSA certificate
LDAP_SSL_DH_3DES_SHA_RSA	"10"	168-bit 3DES encryption with SHA-1 digest and fixed Diffie-Hellman key exchange using RSA certificate
LDAP_SSL_EDH_DES_SHA_DSS_EX	"12"	56-bit DES encryption with SHA-1 digest and ephemeral Diffie-Hellman key exchange using DSS certificate
LDAP_SSL_EDH_3DES_SHA_DSS	"13"	168-bit 3DES encryption with SHA-1 digest and ephemeral Diffie-Hellman key exchange using DSS certificate
LDAP_SSL_EDH_DES_SHA_RSA_EX	"15"	56-bit DES encryption with SHA-1 digest and ephemeral Diffie-Hellman key exchange using RSA certificate
LDAP_SSL_EDH_3DES_SHA_RSA	"16"	168-bit 3DES encryption with SHA-1 digest and ephemeral Diffie-Hellman key exchange using RSA certificate
LDAP_SSL_RSA_AES_128_SHA	"2F"	128-bit AES encryption with SHA-1 digest and RSA key exchange
LDAP_SSL_DH_AES_128_SHA_DSS	"30"	128-bit AES encryption with SHA-1 digest and fixed Diffie-Hellman key exchange using DSS certificate
LDAP_SSL_DH_AES_128_SHA_RSA	"31"	128-bit AES encryption with SHA-1 digest and fixed Diffie-Hellman key exchange using RSA certificate
LDAP_SSL_EDH_AES_128_SHA_DSS	"32"	128-bit AES encryption with SHA-1 digest and ephemeral Diffie-Hellman key exchange using DSS certificate
LDAP_SSL_EDH_AES_128_SHA_RSA	"33"	128-bit AES encryption with SHA-1 digest and ephemeral Diffie-Hellman key exchange using RSA certificate
LDAP_SSL_RSA_AES_256_SHA	"35"	256-bit AES encryption with SHA-1 digest and RSA key exchange
LDAP_SSL_DH_AES_256_SHA_DSS	"36"	256-bit AES encryption with SHA-1 digest and fixed Diffie-Hellman key exchange using DSS certificate
LDAP_SSL_DH_AES_256_SHA_RSA	"37"	256-bit AES encryption with SHA-1 digest and fixed Diffie-Hellman key exchange using RSA certificate
LDAP_SSL_EDH_AES_256_SHA_DSS	"38"	256-bit AES encryption with SHA-1 digest and ephemeral Diffie-Hellman key exchange using DSS certificate
LDAP_SSL_EDH_AES_256_SHA_RSA	"39"	256-bit AES encryption with SHA-1 digest and ephemeral Diffie-Hellman key exchange using RSA certificate

- **LDAP_OPT_SSL_CIPHER_EXPANDED**

  This option is pertinent provided 4-byte SSL ciphers are currently in effect, which is based on the setting of LDAP_OPT_SSL_CIPHER_FORMAT.

  The LDAP_OPT_SSL_CIPHER_EXPANDED option returns a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable. The string consists of a single cipher specification if an SSL connection is established with the LDAP server and provided 4-byte SSL ciphers are in effect. Otherwise, the string consists of one or more cipher suites to be used when negotiating an SSL connection with the LDAP server. The return value is NULL if an SSL connection is not opened and no cipher suites are set by the application. The returned character string consists of the cipher suites that are specified as four hexadecimal digits per cipher suite. Cipher suite values are concatenated, with no separators. The application should call the **ldap_memfree()** routine to release the string when it is no longer needed.

- **LDAP_OPT_SSL_CIPHER_FORMAT**

  The LDAP_OPT_SSL_CIPHER_FORMAT option specifies the cipher format that is used for specifying SSL cipher suites.

  A value of LDAP_SSL_CIPHER_FORMAT_CHAR2 indicates the cipher suites come from either SSL defaults, as determined from the GSK_V3_CIPHER_SPECS environment variable, or from a setting of the LDAP_OPT_SSL_CIPHER by using the **ldap_set_option()** routine.

  A value of LDAP_SSL_CIPHER_FORMAT_CHAR4 indicates the cipher suites come from either SSL defaults, as determined from the GSK_V3_CIPHER_SPECS_EXPANDED environment variable, or from a setting of the LDAP_OPT_SSL_CIPHER_EXPANDED by using the **ldap_set_option()** routine.

- **LDAP_OPT_SSL_TIMEOUT**

  The LDAP_OPT_SSL_TIMEOUT option returns the SSL session timeout value in seconds. Cached SSL sessions are discarded after this number of seconds. Cached SSL sessions can be reused and improve performance by eliminating the need for a full SSL handshake when reconnecting to an LDAP server. The session timeout is 0 if an SSL connection has not been opened and an SSL timeout value has not been set by the application.

- **LDAP_OPT_TIMELIMIT**

  The LDAP_OPT_TIMELIMIT option specifies the number of seconds to wait for search results. The LDAP server can also provide a limit on the search time. For information about the server's search time limit and how it interacts with the client time limit, see the documentation for your LDAP server. For the IBM Tivoli Directory Server for z/OS, see the description of the **timeLimit** configuration file option (Customizing the LDAP server configuration) in *z/OS IBM Tivoli Directory Server Administration and Use for z/OS*. The default time limit for the client, which is specified by a value of 0, indicates that there is no client time limit and that the maximum number of seconds is limited only by the LDAP server limit.

- **LDAP_OPT_UTF8_IO**

  The LDAP_OPT_UTF8_IO option applies to all LDAP API routines that accept an LDAP handle as an input parameter unless stated otherwise in the description of the API routine. Text data for LDAP API routines that do not accept an LDAP handle as an input parameter is in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable.

  The LDAP_OPT_UTF8_IO option returns the format of text data that is provided as input to an LDAP API routine or returned as output by an LDAP API routine.

The return value is LDAP_OPT_ON if text data is in the UTF-8 code set, and LDAP_OPT_OFF if text data is in the code set of the current locale.

- **LDAP_OPT_V2_WIRE_FORMAT**

  The LDAP_OPT_V2_WIRE_FORMAT option returns the format of attribute values that are exchanged between the LDAP client and the LDAP server using the LDAP version 2 protocol. (Attribute values that are exchanged using the LDAP version 3 protocol are always in UTF-8.) The return value is LDAP_OPT_V2_WIRE_FORMAT_ISO8859_1 if attribute values are exchanged using the ISO8859-1 code page. The return value is LDAP_OPT_V2_WIRE_FORMAT_UTF8 if attribute values are exchanged using UTF-8.

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes that are listed in the ldap.h include file.

The following are some common errors for this routine:

LDAP_NO_MEMORY
: Insufficient storage is available.

LDAP_PARAM_ERROR
: A parameter is not valid or the LDAP protocol version is not correct for the requested option.

ldap_get_values()

Purpose

Return the attribute values as an array of character strings

Format

```
#include <ldap.h>

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

Parameters

Input

ld Specifies the LDAP handle.

entry
Specifies an entry returned by the **ldap_first_entry()** or **ldap_next_entry()** routine.

attr
Specifies the attribute type as a null-terminated character string. The string is in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle.

Usage

The **ldap_get_values()** routine returns the attribute values associated with an attribute type as an array of character strings. The attribute type can be supplied by the application or can be an attribute type returned by the **ldap_first_attribute()** or **ldap_next_attribute()** routine.

The attribute values must consist of valid character data, otherwise the results are unpredictable. Use the **ldap_get_values_len()** routine to get binary attribute values.

Function return value

The function return value is an array of character strings, terminated by a NULL string address. Each character string is in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. The application should call the **ldap_value_free()** routine to release the attribute values when they are no longer needed. The return value is NULL if the attribute is not found or if an error is detected. The **ldap_get_errno()** routine can be called to get the error code if the return value is NULL.

The following are some common errors for this routine:

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_NO_SUCH_ATTRIBUTE
Attribute not found.

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_get_values_len()

Purpose

Return the attribute values as an array of binary values

Format

```
#include <ldap.h>

BerVal ** ldap_get_values_len(
    LDAP *               ld,
    LDAPMessage *        entry,
    const char *         attr)
```

Parameters

Input

ld Specifies the LDAP handle.

entry
Specifies an entry returned by the **ldap_first_entry()** or **ldap_next_entry()** routine.

attr
Specifies the attribute type as a null-terminated character string. The string is in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle.

Usage

The **ldap_get_values_len()** routine returns the attribute values associated with an attribute type as an array of binary values. No code page translations are performed on the values. The attribute type can be supplied by the application or can be an attribute type returned by the **ldap_first_attribute()** or **ldap_next_attribute()** routine.

Function return value

The function return value is an array of binary values. The array is terminated by a NULL BerVal address. The application should call the **ldap_value_free_len()** routine to release the attribute values when they are no longer needed. The return value is NULL if the attribute is not found or if an error is detected. The **ldap_get_errno()** routine can be called to get the error code.

The following are some common errors for this routine:

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_NO_SUCH_ATTRIBUTE
Attribute not found.

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_init()

Purpose

Create and initialize an LDAP handle for an SSL or non-SSL connection

Format

```
#include <ldap.h>

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

Parameters

Input

host
Specifies the location of the LDAP server as a null-terminated string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable. This location can be a blank-separated host list or a single LDAP URL. Specify NULL for this parameter to connect to an LDAP server on the local system using the IPv4 loopback address (127.0.0.1).

port
Specifies the port for the LDAP server when an explicit port is not specified in the host list. The value must be between 1 and 65535. Specify 0 to use the default LDAP port (389).

Usage

The **ldap_init()** routine creates and initializes an LDAP handle. The routine does not establish a connection with the LDAP server. A connection is established when the first server request using the handle is issued. The handle is initialized for a non-SSL connection unless an LDAP URL is specified for the host parameter and the URL scheme is **ldaps** instead of **ldap**. The application should call the **ldap_unbind()** or **ldap_unbind_s()** routine to release the handle when it is no longer needed. The location of the LDAP server can be explicitly specified by using a host list or an LDAP URL containing a host name. The location of the LDAP server can be implicitly specified by using an LDAP URL that does not contain a host name.

A host list consists of one or more blank-separated *host:port* values. The host specification is a DNS resource name (for example, dcesec4.endicott.ibm.com), a dotted decimal IPv4 address (for example, 9.130.25.34), or a colon-separated IPv6 address enclosed in square brackets (for example, [1080::8:800:200C:417A]. The port, if specified, must be a decimal number between 1 and 65535. The value of the *port* parameter can be used if a port is not specified. The hosts are tried in the order specified until a connection is established with an LDAP server.

An LDAP URL has the following format:

[**<**][**URL:**]*scheme*[**://**][*host*[**:***port*]][*/dn*[**?***attributes*[**?***scope*[**?***filter*]]]][**>**]

where:

scheme
Specifies the value **ldap** for a non-SSL connection and **ldaps** for an SSL connection.

host:port
Specifies the location of the LDAP server. The host specification can be a DNS resource name (for example, dcesec4.endicott.ibm.com), a dotted decimal IPv4 address (for example, 9.130.25.34), or a colon-separated IPv6 address enclosed in square brackets (for example, [1080::8:800:200C:417A]). The port, if specified, must be a decimal number between 1 and 65535. The port defaults to 389 for a non-SSL connection and 636 for an SSL connection.

dn Specifies the distinguished name (DN) for the request. The DN can be used as a filter when the **ldap_server_locate()** routine should be called to locate the LDAP server.

attributes
Consists of one or more comma-separated search attributes. This value is not used by the **ldap_init()** routine.

scope Specifies the search scope and can be "base", "one", or "sub". This value is not used by the **ldap_init()** routine.

filter Specifies the search filter. This value is not used by the **ldap_init()** routine.

The URL can be optionally enclosed in angle brackets or prefixed with URL: or both.

The **ldap_init()** routine calls the **ldap_server_locate()** routine to locate the LDAP server when the LDAP URL does not contain a host name. The default server information file /etc/ldap/ldap_server_info.conf can be used unless the LDAP_SERVER_INFO_CONF environment variable is defined. The **ldap_server_locate()** routine uses the default values for everything except the DN filter. The DN filter is set to the DN specified in the URL. (No DN filtering is done if a DN is not specified in the URL.) The scheme specified in the URL can be used to select servers from the list returned by the **ldap_server_locate()** routine. A server entry is selected if the scheme is **ldap** and the security type is LDAP_LSI_NOSSL or if the scheme is **ldaps** and the security type is LDAP_LSI_SSL. A server entry is not selected if the security type is not defined.

The **ldap_ssl_client_init()** routine must be called before the **ldap_init()** routine if the LDAP URL specifies an SSL connection.

The LDAP handle is initialized with the following default values. The **ldap_set_option()** or **ldap_set_option_np()** routine can be called to set different values upon completion of the **ldap_init()** routine.

- The LDAP protocol version is set based on the LDAP_VERSION environment variable. If the LDAP_VERSION environment variable is not defined, the protocol version is set to 3.
- The LDAP version 2 wire format is set based on the LDAP_V2_WIRE_FORMAT environment variable. If the LDAP_V2_WIRE_FORMAT environment variable is not defined, the LDAP version 2 wire format is set to UTF-8.
- Referral processing is enabled and the referral hop limit is set to 10.

Function return value

The function return value is the new LDAP handle if no error is detected. Otherwise, the return value is `NULL`.

ldap_insert_control()

Purpose

Insert an existing control into a list of controls

Format

```
#include <ldap.h>

int ldap_insert_control(
    LDAPControl *           control,
    LDAPControl ***         control_list)
```

Parameters

Input

control
Specifies the control to be added to the list of controls.

Output

control_list
Specifies the address of the control list. A new control list is created if there is no control list. (The location pointed to by the *control_list* parameter contains NULL.) Otherwise, the existing control list is expanded and the new control is added to the list. The **ldap_controls_free()** routine should be called to release the controls when they are no longer needed.

Usage

The **ldap_insert_control()** routine adds an existing control to a list of controls. The control list is reallocated to make room for the new control.

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, the return value is one of the LDAP error codes listed in the ldap.h include file.

The following are some common client errors:

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_is_ldap_url()

Purpose

Determine if a URL appears to be an LDAP URL

Format

```
#include <ldap.h>

int ldap_is_ldap_url(
    const char *            url)
```

Parameters

Input

url
Specifies the URL to be tested as a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable

Usage

The **ldap_is_ldap_url()** routine checks the supplied URL to see if it looks like an LDAP URL. An LDAP URL has the following format:

[**<**][**URL:**]*scheme*[**://**][*host*[**:***port*]][*/dn*[**?***attributes*[**?***scope*[**?***filter*]]]][**>**]

The **ldap_is_ldap_url()** routine checks for a scheme of **ldap** or **ldaps**. The routine does not check the remainder of the URL. To validate the entire URL, use the **ldap_url_parse()** routine instead of the **ldap_is_ldap_url()** routine.

Function return value

If the URL is an LDAP URL, the function return value is 1 (TRUE). If it is not, the return value is 0 (FALSE).

ldap_is_ldap_url_np()

Purpose

Determine if a URL appears to be an LDAP URL

Format

```
#include <ldap.h>

int ldap_is_ldap_url_np(
    LDAP *                ld,
    const char *          url)
```

Parameters

Input

ld Specifies an LDAP handle. If the URL is in UTF-8, this parameter can be specified as NULL. Otherwise, the URL is in either the local EBCDIC code page or UTF-8 as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle.

url
Specifies the URL to be tested as a null-terminated character string in either the local EBCDIC code page or UTF-8 as determined by the LDAP handle.

Usage

The **ldap_is_ldap_url_np()** routine is the same as the **ldap_is_ldap_url()** routine except that the URL is in either UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option. For information about the **ldap_is_ldap_url()** routine, see "ldap_is_ldap_url()" on page 111.

Function return value

If the URL is an LDAP URL, the function return value is 1 (TRUE). If it is not, the return value is 0 (FALSE).

ldap_memcache_destroy()

Purpose

Destroy a search result cache

Format

```
#include <ldap.h>

void ldap_memcache_destroy(
    LDAPMemCache *         cache)
```

Parameters

Input

cache
Specifies the search result cache handle.

Usage

The **ldap_memcache_destroy()** routine destroys a search result cache created by the **ldap_memcache_init()** routine. The cache handle is not valid upon completion of this routine. Search result caching is disabled for any LDAP handles that are still associated with the search result cache.

The global search result cache cannot be destroyed. If the routine should be called with the cache handle for the global search result cache, all entries in the global cache are removed but the global cache remains valid.

Function return value

There is no function return value.

ldap_memcache_flush()

Purpose

Remove entries from a search result cache

Format

```
#include <ldap.h>

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

Parameters

Input

cache
Specifies the search result cache handle.

dn Specifies the base distinguished name as a null-terminated character string in UTF-8 or the local EBCDIC code page, as determined by the LDAP_LIBASCII compiler variable. Specify NULL for this parameter to flush all cache entries.

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

LDAP_SCOPE_BASE
Search just the entry specified by the base name.

LDAP_SCOPE_ONELEVEL
Search the base entry and its immediate children.

LDAP_SCOPE_SUBTREE
Search the base entry and all of its descendants.

Usage

The **ldap_memcache_flush()** routine removes entries from a search result cache. The *dn* parameter specifies the base distinguished name and the *scope* parameter specifies the name scope. All search requests whose base distinguished names fall within the range of the specified DN and scope are removed from the cache.

Examples: Assume that the cache contains search requests for the following base distinguished names:

```
o=Acme
ou=Manufacturing,o=Acme
ou=Research,o=Acme
cn=John Doe,ou=Manufacturing,o=Acme
cn=Jane Doe,ou=Research,o=Acme
```

- If **ldap_memcache_flush()** should be called with "o=Acme" and scope=LDAP_SCOPE_BASE, the "o=Acme" cache entry is removed.
- If **ldap_memcache_flush()** should be called with "o=Acme" and scope=LDAP_SCOPE_ONELEVEL, the "ou=Manufacturing,o=Acme" and "ou=Research,o=Acme" entries are removed.
- If **ldap_memcache_flush()** should be called with "o=Acme" and scope=LDAP_SCOPE_SUBTREE, all entries are removed.

Function return value

There is no function return value.

ldap_memcache_get()

Purpose

Return the search result cache for an LDAP handle

Format

```
#include <ldap.h>

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

Parameters

Input

ld Specifies the LDAP handle.

Output

cachep
Returns the cache handle. If there is no search result cache for the LDAP handle, the return value is NULL.

Usage

The **ldap_memcache_get()** routine returns the search result cache handle associated with the LDAP handle.

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes listed in the ldap.h include file.

The following is a common error for this routine:

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_memcache_init()

Purpose

Create a search result cache

Format

```
#include <ldap.h>

int ldap_memcache_init(
    unsigned long              ttl,
    unsigned long              size,
    char *                     baseDNs[],
    void *                     reserved,
    LDAPMemCache **            cachep)
```

Parameters

Input

ttl
Specifies the lifetime in seconds for entries in the cache. The maximum value is 2147483647 seconds. Specify 0 if the cache entries do not expire.

size
Specifies the maximum size in bytes for the cache. The maximum value is 2147483647 bytes. Specify 0 if there is no maximum size for the cache. Older entries are removed to make room for new entries once the maximum size is reached.

baseDNs
Specifies an array of distinguished names. The end of the array is indicated by a NULL address. Each distinguished name is a null-terminated character string in UTF-8 or the local EBCDIC code page, as determined by the LDAP_LIBASCII compiler variable. Specify NULL for this parameter to cache all search results.

reserved
Specify NULL for this parameter.

Output

cachep
Returns the cache handle. The **ldap_memcache_destroy()** routine should be called to destroy the cache when it is no longer needed.

Usage

The **ldap_memcache_init()** routine creates a search result cache. The *baseDNs* parameter specifies the list of base distinguished names. The search request is not cached if the base DN for the search request is not included in this list. All search requests are cached if NULL is specified for the *baseDNs* parameter.

After the search result cache is created, the **ldap_memcache_set()** routine must be called to associate the search result cache with one or more LDAP handles. Search requests using these LDAP handles are then cached in the search result cache.

The LDAP_CLIENT_CACHE environment variable can be used to define a global search result cache. All LDAP handles use the global search result cache unless the

ldap_memcache_set() routine should be called to set a different cache for the LDAP handle.

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes listed in the ldap.h include file.

The following are some common errors for this routine:

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_memcache_set()

Purpose

Set the search result cache for an LDAP handle

Format

```
#include <ldap.h>

int ldap_memcache_set(
    LDAP *                  ld,
    LDAPMemCache *          cache)
```

Parameters

Input

ld Specifies the LDAP handle.

cache
Specifies the search result cache handle. Specify `NULL` for this parameter to disable search result caching for the LDAP handle.

Usage

The **ldap_memcache_set()** routine sets the search result cache used by the LDAP handle. The **ldap_memcache_init()** routine can be used to create a search result cache.

Function return value

The function return value is `LDAP_SUCCESS` if no error is detected. Otherwise, it is one of the LDAP error codes listed in the `ldap.h` include file.

The following are some common errors for this routine:

LDAP_LOCAL_ERROR
An error is detected by a system routine.

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_memcache_update()

Purpose

Remove expired search result cache entries

Format

```
#include <ldap.h>

void ldap_memcache_update(
    LDAPMemCache *         cache)
```

Parameters

Input

cache
Specifies the search result cache handle.

Usage

The **ldap_memcache_update()** routine removes all expired entries from the search result cache. It is normally not necessary to call the **ldap_memcache_update()** routine, because expired cache entries are automatically removed when new entries are added to the cache.

Function return value

There is no function return value.

ldap_memfree()

Purpose

Release storage allocated by the LDAP run time

Format

```
#include <ldap.h>

void ldap_memfree(
    void *                  mem)
```

Parameters

Input

mem
Specifies the address of the storage to be released.

The **ldap_memfree()** routine releases storage allocated by the LDAP run time.

Function return value

There is no function return value.

ldap_modify(), ldap_modify_s(), ldap_modify_ext(), ldap_modify_ext_s()

Purpose

Modify an existing entry in the LDAP directory

Format

```
#include <ldap.h>

typedef struct ldapmod {
    int                         mod_op;
    char *                      mod_type;
    union {
        char **                 modv_strvals;
        BerVal **               modv_bvals;
    } mod_vals;
    struct ldapmod *            mod_next;
} LDAPMod;

#define LDAP_MOD_BVALUES        0x80

#define mod_values              mod_vals.modv_strvals
#define mod_bvalues             mod_vals.modv_bvals

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

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

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

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

Parameters

Input

ld Specifies the LDAP handle.

dn Specifies the distinguished name for the directory entry as a null-terminated character string in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. A zero-length name is not allowed for a modify request.

mods
Specifies the attribute modifications for the directory entry. The *mod_op* field indicates whether the LDAP server should add the attribute (LDAP_MOD_ADD), replace the attribute (LDAP_MOD_REPLACE) or delete the attribute

(LDAP_MOD_DELETE). The LDAP_MOD_BVALUES flag should be set in the *mod_op* field for binary attribute values. The *mod_type* field specifies the attribute type as a null-terminated character string in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. The *modv_strvals* field can be used for character values, while the *modv_bvals* field can be used for binary values. The supplied values are in binary if the LDAP_MOD_BVALUES flag is set. Otherwise, the supplied values are null-terminated character strings in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle.

serverctrls
Specifies an array of server controls for the add request. The end of the array is indicated by a NULL address. If NULL is specified for this parameter, the server controls specified by the LDAP_OPT_SERVER_CONTROLS option for the LDAP handle are used. If NULL is specified for this parameter and the LDAP_OPT_SERVER_CONTROLS option has not been set for the LDAP handle, no server controls are used. The server controls for the LDAP handle can be overridden so that no controls are used by specifying a server controls array consisting of a NULL address. (Control values for this routine vary depending on whether you are specifying server or client controls. See "LDAP controls" on page 15 for details.)

clientctrls
Specifies an array of client controls for the add request. The end of the array is indicated by a NULL address. If NULL is specified for this parameter, the client controls specified by the LDAP_OPT_CLIENT_CONTROLS option for the LDAP handle are used. If NULL is specified for this parameter and the LDAP_OPT_CLIENT_CONTROLS option has not been set for the LDAP handle, no client controls are used. The client controls for the LDAP handle can be overridden so that no controls are used by specifying a client controls array consisting of a NULL address. (Control values for this routine vary depending on whether you are specifying server or client controls. See "LDAP controls" on page 15 for details.)

Output

msgidp
Returns the message identifier assigned to the modify request message. This value can be used when calling the **ldap_result()** routine to wait for the modify result message.

Usage

The **ldap_modify()** and **ldap_modify_ext()** routines send the request to the LDAP server and return control to the application. The application must call the **ldap_result()** routine to obtain the result.

The **ldap_modify_s()** and **ldap_modify_ext_s()** routines send the request to the LDAP server and wait for the completion of the request. The modify request is abandoned if the client is unable to wait for the response because of an error from the **ldap_result()** routine.

The entry to be modified must exist. The modifications are performed as an atomic unit in the order listed and either all the modifications are performed or none of the modifications are performed. The directory schema can be violated while the modifications are performed, but the final result must conform to the requirements of the directory schema. If the z/OS LDAP server is running with an SDBM

backend, the **ldap_modify()** APIs can return the LDAP_OTHER error code and have completed a partial update to an entry in RACF. The results match what would occur if the update was done using the RACF **ALTUSER, ALTGROUP**, or **RALTER** command. The RACF message text is also returned in the result.

To add attribute values, set the *mod_op* field to LDAP_MOD_ADD. Existing attribute values remain unchanged. The attribute is created if it does not exist.

To replace attribute values, set the *mod_op* field to LDAP_MOD_REPLACE. When modifying directory entries, you must specify the entire set of attribute values. Any existing attribute values not included in the replacement are removed. The attribute is created if it does not exist. The attribute is deleted if no attribute values are specified.

When modifying a schema on a z/OS LDAP server, you can replace an attribute value without specifying all the other values in the set. A value is replaced if it exists in the schema attribute. An attribute is added if it does not exist in the schema attribute. No attributes are removed.

To delete attribute value, set the *mod_op* field to LDAP_MOD_DELETE. The supplied values are removed from the attribute. All attribute values are deleted if no values are provided. The attribute is deleted if there are no values left after deleting the requested values.

The attributes making up the low-level RDN of the distinguished name for the entry cannot be modified. However, if these attributes are multi-valued, other (non-RDN) values can be added or removed. Use the **ldap_rename()** or **ldap_rename_s()** routine to change the entry name.

Mandatory attributes for the entry object classes cannot be removed. Any mandatory attributes required by new object classes that are added to the entry must be added as part of the same modify operation.

The **ldap_modify()** and **ldap_modify_s()** routines use client controls specified by the LDAP_OPT_CLIENT_CONTROLS and server controls specified by the LDAP_OPT_SERVER_CONTROLS options. The **ldap_modify_ext()** and **ldap_modify_ext_s()** routines also use these controls unless overridden by the *serverctrls* and *clientctrls* parameters.

Function return value

The **ldap_modify()** routine returns -1 if a client error is detected. Otherwise, it returns the message identifier assigned to the modify request. The application should call the **ldap_get_errno()** routine to get the error code if the return value is -1. The **ldap_modify()** routine does not return errors reported by the LDAP server. Instead, the application must call the **ldap_parse_result()** routine to obtain the result code from the result message returned by the **ldap_result()** routine.

The **ldap_modify_ext()** routine returns LDAP_SUCCESS if the request is sent to the LDAP server. Otherwise, the return value is one of the error codes listed in the ldap.h include file. The **ldap_modify_ext()** routine does not return errors reported by the LDAP server. Instead, the application must call the **ldap_parse_result()** routine to obtain the result code from the result message returned by the **ldap_result()** routine.

The **ldap_modify_s()** and **ldap_modify_ext_s()** routines return LDAP_SUCCESS if the request is successful. Otherwise, the return value is one of the error codes listed in the ldap.h include file. The return value includes errors detected by the LDAP client and errors detected by the LDAP server.

The following are some common client errors:

LDAP_INVALID_STATE
An unbind request has been issued for the LDAP handle.

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_NOT_SUPPORTED
The LDAP protocol version must be LDAP_VERSION3 to specify server or client controls.

LDAP_PARAM_ERROR
A parameter is not valid.

LDAP_SERVER_DOWN
Network connection failed.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION
A critical client control is either not recognized or is not supported for a modify operation.

The following are some common server result codes:

LDAP_INSUFFICIENT_ACCESS
Not authorized to modify entry.

LDAP_NO_SUCH_OBJECT
The entry does not exist.

LDAP_OBJECT_CLASS_VIOLATION
Either a mandatory attribute is not included or an attribute is not allowed by the object class definition.

LDAP_REFERRAL
The entry is not in the current LDAP server.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION
A critical server control is either not recognized or is not supported for a modify operation.

LDAP_UNDEFINED_TYPE
An attribute type is not defined in the directory schema.

ldap_mods_free()

Purpose

Release storage allocated for an array of attribute modifications

Format

```
#include <ldap.h>

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

Parameters

Input

mods
Specifies the array of attribute modifications. The end of the array is indicated by a NULL address.

freemods
Specify TRUE(1) to free the LDAPMod address array and the individual LDAPMod structures. Specify FALSE(0) to free only the individual LDAPMod structures.

Usage

The **ldap_mods_free()** routine releases the storage allocated for an array of attribute modifications. The attribute type and value are released along with the LDAPMod structure. If a nonzero value is specified for the *freemods* parameter, the LDAPMod address array is freed as well.

Function return value

There is no function return value.

ldap_msgfree()

Purpose

Release storage for an LDAP message

Format

```
#include <ldap.h>

int ldap_msgfree(
    LDAPMessage *              msg)
```

Parameters

Input

msg
Specifies the LDAP message to be released.

Usage

The **ldap_msgfree()** routine releases the storage allocated for an LDAP message and its message chain.

Function return value

The function return value is the message type of the message. If there is a message chain, the function return value is the message type of the last message in the chain. The function return value is `0` if the message address is `NULL` or is not the address of an LDAP message.

ldap_msgid()

Purpose

Return the message identifier

Format

```
#include <ldap.h>

int ldap_msgid(
    LDAPMessage *               msg)
```

Parameters

Input

msg
Specifies the LDAP message.

Usage

The **ldap_msgid()** routine returns the message identifier for an LDAP message.

Function return value

The function return value is the message identifier. The function return value is `0` if the message address is `NULL` or is not the address of an LDAP message.

ldap_msgtype()

Purpose

Return the message type

Format

```
#include <ldap.h>

int ldap_msgtype(
    LDAPMessage *                msg)
```

Parameters

Input

msg
Specifies the LDAP message.

Usage

The **ldap_msgtype()** routine returns the message type for an LDAP message.

Function return value

The function return value is the message type. If the message address is `NULL` or is not the address of an LDAP message, the function return value is `0`.

ldap_next_attribute()

Purpose

Return the attribute type for the next attribute in an LDAP search entry

Format

```
#include <ldap.h>

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

Parameters

Input

ld Specifies the LDAP handle.

entry
Specifies an entry returned by the **ldap_first_entry()** or **ldap_next_entry()** routine.

ber
Specifies the LDAP control block returned by the **ldap_first_attribute()** routine.

Usage

The **ldap_next_attribute()** routine returns the attribute type for the next attribute in the search entry. The **ldap_get_values()** or **ldap_get_values_len()** routine can then be called to get the attribute values associated with the attribute type.

The *ber* parameter is a control block allocated by the **ldap_first_attribute()** routine and maintained by the LDAP client runtime. This control block is released when the **ldap_next_attribute()** routine returns a NULL value, even if the NULL value is the result of an error. The application should call the **ldap_memfree()** routine to release this control block if the application does not want to keep calling the **ldap_next_attribute()** routine until all attributes have been processed.

Function return value

The function return value is the attribute type of the next attribute. The attribute type is a null-terminated character string in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. The application should call the **ldap_memfree()** routine to release the attribute type when it is no longer needed. The return value is NULL if there are no more attributes or if an error is detected. The **ldap_get_errno()** routine can be called to get the error code when the return value is NULL. The error code is LDAP_SUCCESS if there are no more attributes.

The following are some common errors for this routine:

LDAP_NO_MEMORY
Insufficient storage available.

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_next_entry()

Purpose

Return the next search entry in an LDAP result

Format

```
#include <ldap.h>

LDAPMessage * ldap_next_entry(
    LDAP *               ld,
    LDAPMessage *        msg)
```

Parameters

Input

ld Specifies the LDAP handle.

msg
Specifies the LDAP message returned by the **ldap_first_entry()** routine.

Usage

The **ldap_next_entry()** routine returns the address of the next search entry in an LDAP result.

Function return value

The function return value is the address of the next search entry. The return value is NULL if there are no more search entries or if an error is detected. The **ldap_get_errno()** routine can be called to get the error code when the return value is NULL. The error code is LDAP_SUCCESS if there are no more search entries.

The following is a common error for this routine:

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_next_message()

Purpose

Return the next LDAP message in an LDAP result

Format

```
#include <ldap.h>

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

Parameters

Input

ld Specifies the LDAP handle.

msg
Specifies the LDAP message returned by the **ldap_first_message()** routine.

Usage

The **ldap_next_message()** routine returns the address of the next message in an LDAP result.

Function return value

The function return value is the address of the next message. The return value is NULL if there are no more messages or if an error is detected. The **ldap_get_errno()** routine can be called to get the error code when the return value is NULL. The error code is LDAP_SUCCESS if there are no more messages.

The following is a common error for this routine:

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_next_reference()

Purpose

Return the next search reference in an LDAP result

Format

```
#include <ldap.h>

LDAPMessage * ldap_next_reference(
    LDAP *               ld,
    LDAPMessage *        msg)
```

Parameters

Input

ld Specifies the LDAP handle.

msg
Specifies the LDAP message returned by the **ldap_first_reference()** routine.

Usage

The **ldap_next_reference()** routine returns the address of the next search reference in an LDAP result.

Function return value

The function return value is the address of the next search reference. The return value is NULL if there are no more search references or if an error is detected. The **ldap_get_errno()** routine can be called to get the error code when the return value is NULL. The error code is LDAP_SUCCESS if there are no more search references.

The following is a common error for this routine:

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_parse_entrychange_control()

Purpose

Parse an entry change notification server control returned in an LDAP search response

Format

```
#include <ldap.h>

int ldap_parse_entrychange_control(
    LDAP *                  ld,
    LDAPControl *           server_controls[],
    int *                   change_type,
    char **                 previous_dn,
    int *                   change_number_present,
    long *                  change_number)
```

Parameters

Input

ld Specifies the LDAP handle.

server_controls
Specifies an array of server controls returned in the response message. The end of the array is indicated by a NULL address. The array should contain an entry change notification control.

Output

change_type
Returns the change type and is LDAP_CHANGETYPE_ADD, LDAP_CHANGETYPE_DELETE, LDAP_CHANGETYPE_MODIFY or LDAP_CHANGETYPE_MODDN. Specify NULL for this parameter if the change type is not needed.

previous_dn
Returns the entry DN before it was renamed or moved by a Modify DN operation and is NULL for other types of changes. Specify NULL for this parameter if the previous DN is not needed. The name is a null-terminated character strong in UTF-8 or the local EBCDIC code page as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. The **ldap_memfree()** routine should be called to release the name when it is no longer needed.

change_number_present
Returns 1 if the change number is returned by the LDAP server or 0 if the change number is not returned. Specify NULL for this parameter if the change number indication is not needed.

change_number
Returns the change number if one was returned by the LDAP server. Specify NULL for this parameter if the change number is not needed.

Usage

The **ldap_parse_entrychange_control()** routine can be used to process the entry change notification control (2.16.840.1.113730.3.4.7) returned by the LDAP server in an LDAP search entry.

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes listed in the ldap.h include file.

The following are some common client errors:

LDAP_CONTROL_NOT_FOUND
The server controls do not contain the entry change notification control.

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_parse_extended_result()

Purpose

Parse an LDAP extended result message

Format

```
#include <ldap.h>

int ldap_parse_extended_result(
    LDAP *                ld,
    LDAPMessage *         result,
    char **               resultoidp,
    BerVal **             resultdatap,
    int                   freeit)
```

Parameters

Input

ld Specifies the LDAP handle.

result
Specifies the result message returned by the **ldap_result()** or **ldap_extended_operation_s()** routines.

freeit
Specify TRUE(1) to free the LDAP message chain before returning to the application or specify FALSE(0) to keep the LDAP message chain. If you specify TRUE, the message chain is freed even when the function return value is not LDAP_SUCCESS.

Output

resultoidp
Returns the response OID from the extended result message. Specify NULL for this parameter if the response OID should not be returned. The value is set to NULL if the LDAP server did not return a response OID. The OID is returned as a null-terminated dotted decimal character string in either UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. The application should call the **ldap_memfree()** routine to release the OID string when it is no longer needed.

resultdatap
Returns the response data from the extended result message. Specify NULL for this parameter if the response data should not be returned. The value is set to NULL if the LDAP server did not return any response data. The application should call the **ldap_berfree_np()** routine to release the response data when it is no longer needed.

Usage

The **ldap_parse_extended_result()** routine returns extended response information from an LDAP extended result message. A parameter error is returned if the message is not an extended result message. The application can call the **ldap_parse_result()** routine to obtain the matched name, error text, and referral information from the extended result message.

Function return value

The function return value is the result code from the extended result message unless an error is detected while parsing the message.

The following are some common errors for this routine:

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_parse_page_control()

Purpose

Parse a paged results server control returned in an LDAP search response

Format

```
#include <ldap.h>

int ldap_parse_page_control(
    LDAP *             ld,
    LDAPControl *      server_controls[],
    unsigned long *    total_count,
    BerVal **          cookie)
```

Parameters

Input

ld Specifies the LDAP handle.

server_controls
Specifies an array of server controls returned in the response message. The end of the array is indicated by a NULL address. The array should contain a paged results server control.

Output

total_count
Returns the server estimate of the total number of entries in the total result set. This value is zero if the server is unable to provide an estimate of the total number of entries.

cookie
Returns the cookie for the next page of search results. The **ldap_berfree_np()** routine should be called to release the *cookie* when it is no longer needed.

Usage

RFC 2696: LDAP Control Extension for Simple Paged Results Manipulation provides paging capabilities for LDAP clients that want to receive just a subset of search results (page) instead of the entire list. The next page of entries is returned to the client application for each subsequent paged results search request submitted by the client until the operation is canceled or the last result is returned. See the description of "ldap_create_page_control()" on page 52 for a detailed description of paged search results processing.

The **ldap_parse_page_control()** routine can be used to extract the total entry count and the *cookie* from the paged results control returned by the LDAP server. The server returns a zero-length *cookie* when the last page of results is returned. The server controls in the search response message do not contain the paged results control if the requested page size is greater than or equal to the size limit in the search request or if there are no result entries to return.

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes listed in the ldap.h include file.

The following are some common client errors:

LDAP_CONTROL_NOT_FOUND
The server controls do not contain the paged results control

LDAP_NO_MEMORY
Insufficient storage is available

LDAP_PARAM_ERROR
A parameter is not valid

ldap_parse_pwdpolicy_response()

Purpose

Parse a password policy control response returned in an LDAP message

Format

```
#include <ldap.h>

int ldap_parse_pwdpolicy_response(
    LDAPControl *          server_controls[],
    int *                  controlerrp,
    int *                  controlwarnp,
    int *                  controlresp)
```

Parameters

Input

server_controls
Specifies an array of server controls returned in the response message. The end of the array is indicated by a NULL address. The array should contain a server password policy control response.

Output

controlerrp
Returns the LDAP password policy error code, that can be used as input to **ldap_pwdpolicy_err2string()** to obtain a text description of the error.

controlwarnp
Returns the LDAP password policy warning code, that can be used as input to **ldap_pwdpolicy_err2string()** to obtain a text description of the warning.

controlresp
Returns the warning result value.

Usage

The **ldap_pwdpolicy_err2string()** routine, given a password policy control response (1.3.6.1.4.1.42.2.27.8.5.1) error or warning code from the **ldap_parse_pwdpolicy_response()** routine, returns a null-terminated character string that provides a textual description of the password policy error or warning.

The **ldap_parse_pwdpolicy_response()** routine is used to:

- Obtain the LDAP password policy error or warning codes from the password policy control response associated with an LDAP message.
- Obtain the LDAP password policy warning result value associated with the warning code from the password policy control response.

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes listed in the ldap.h include file.

The following are some common client errors:

LDAP_CONTROL_NOT_FOUND
The server controls do not contain the password policy control response.

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_parse_reference_np()

Purpose

Parse an LDAP search continuation reference message

Format

```
#include <ldap.h>

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

Parameters

Input

ld Specifies the LDAP handle.

result
Specifies the result message returned by **ldap_result()** or one of the synchronous search request routines.

freeit
Specify `TRUE(1)` to free the LDAP message chain before returning to the application. If you specify `TRUE`, the message chain is freed even when the function return value is not `LDAP_SUCCESS`. Specify `FALSE(0)` to keep the LDAP message chain.

Output

referralsp
Returns the referrals as an array of character strings. The end of the array is indicated by a `NULL` string address. The return value is `NULL` if the LDAP server did not return any referrals. Specify `NULL` for this parameter if the referral list should not be returned. Each referral is returned as a null-terminated character string in either UTF-8 or the local EBCDIC code page, as determined by the `LDAP_OPT_UTF8_IO` option for the LDAP handle. The application should call the **ldap_value_free()** routine to release the referrals array when it is no longer needed.

serverctrlsp
Returns the server controls as an array of `LDAPControl` structures. The end of the array is indicated by a `NULL` control address. The return value is `NULL` if the LDAP server did not return any server controls. Specify `NULL` for this parameter if the server controls should not be returned. The control OID string is in UTF-8 or the local EBCDIC code page, as determined by the `LDAP_OPT_UTF8_IO` option for the LDAP handle. The control value is unchanged and has the format returned by the LDAP server. The application should call the **ldap_controls_free()** routine to release the controls array when it is no longer needed. (Control values for this routine vary depending on whether you are specifying server or client controls. See “LDAP controls” on page 15 for details.)

Usage

The **ldap_parse_reference_np()** routine returns information from a search continuation reference message. It returns a parameter error if the message is not a search continuation reference message.

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes listed in the ldap.h include file.

The following are some common errors for this routine:

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_parse_result()

Purpose

Parse an LDAP result message

Format

```
#include <ldap.h>

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

Parameters

Input

ld Specifies the LDAP handle.

result
Specifies the result message returned by **ldap_result()** or one of the synchronous request routines.

freeit
Specify TRUE(1) to free the LDAP message chain before returning to the application. If you specify TRUE, the message chain is freed even when the function return value is not LDAP_SUCCESS. Specify FALSE(0) to keep the LDAP message chain.

Output

errcodep
Returns the result code from the result message. Specify NULL for this parameter if the result code should not be returned.

matcheddnp
Returns the matched distinguished name from the result message as a null-terminated character string. The string is in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. The return value is NULL if the result message does not contain a matched distinguished name. The application should call the **ldap_memfree()** routine to release the string when it is no longer needed. Specify NULL for this parameter if the matched distinguished name should not be returned.

errmsgp
Returns the error text from the result message as a null-terminated character string. The string is in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. The return value is NULL if the LDAP server did not return any error text. The application should call the **ldap_memfree()** routine to release the string when it is no longer needed. Specify NULL for this parameter if the error text should not be returned.

referralsp
Returns the referrals as an array of null-terminated character strings. The end of the array is indicated by a NULL string address. The strings are in UTF-8 or

the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. The return value is NULL if the LDAP server did not return any referrals. The application should call the **ldap_value_free()** routine to release the referrals array when it is no longer needed. Specify NULL for this parameter if the referrals should not be returned.

servctrlsp
Returns the server controls as an array of LDAPControl structures. The end of the array is indicated by a NULL control address. The return value is NULL if the LDAP server did not return any server controls. The control OID string is in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. The control value is unchanged and has the format returned by the LDAP server. The application should call the **ldap_controls_free()** routine to release the controls array when it is no longer needed. Specify NULL for this parameter if the server controls should not be returned.

Usage

The **ldap_parse_result()** routine returns information from an LDAP result message. The routine returns an error if it should be called for a search entry or search reference message and the message chain does not contain the search result message. The application can obtain additional information from a SASL bind result message by calling the **ldap_parse_sasl_bind_result()** routine. The application can obtain additional information from an extended result message by calling the **ldap_parse_extended_result()** routine.

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes listed in the ldap.h include file.

The following are some common errors for this routine:

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_NO_RESULT_MESSAGE
The message chain does not contain an LDAP result.

LDAP_PARAM_ERROR
A parameter is not valid.

Note: Before z/OS V1R6, **ldap_parse_result()** returned LDAP_OPERATIONS_ERROR when it was called to process a search entry or search reference message and the message chain did not contain the search result message. As of z/OS V1R6, the LDAP client returns LDAP_NO_RESULT_MESSAGE.

ldap_parse_sasl_bind_result()

Purpose

Parse an LDAP SASL bind result message

Format

```
#include <ldap.h>

int ldap_parse_sasl_bind_result(
    LDAP *                ld,
    LDAPMessage *         result,
    BerVal **             servercredp,
    int                   freeit)
```

Parameters

Input

ld Specifies the LDAP handle.

result
Specifies the result message returned by the **ldap_result()** routine.

freeit
Specify TRUE to free the result message before returning to the application. If you specify TRUE, the result message is freed even when the function return value is not LDAP_SUCCESS. Specify FALSE to keep the result message.

Output

servercredp
Returns the server credentials from the result message. The return value is NULL if there are no server credentials. Specify NULL for this parameter if the server credentials should not be returned. The application should call the **ldap_berfree_np()** routine to release the credentials when they are no longer needed.

Usage

The **ldap_parse_sasl_bind_result()** routine returns information from a SASL (Simple Authentication and Security Layer) bind result message.

Function return value

The function return value is the result code from the bind result message unless an error is detected while parsing the message.

The following are some common errors for this routine:

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_parse_sort_control()

Purpose

Parse a sort results response control returned in an LDAP search response

Format

```
#include <ldap.h>

int ldap_parse_sort_control(
    LDAP *                  ld,
    LDAPControl *           server_controls[],
    unsigned long *         sort_rc,
    char **                 attribute)
```

Parameters

Input

ld Specifies the LDAP handle.

server_controls
Specifies an array of server controls returned in the response message. The end of the array is indicated by a NULL address. The array should contain a sort results server control.

Output

sort_rc
Returns the sort result code.

attribute
Returns the attribute name associated with a sort error. Specify NULL for this parameter if the attribute name is not needed. The return value is NULL if the server did not return an attribute name. The **ldap_memfree()** routine should be called to release the attribute name when it is no longer needed.

Usage

RFC 2891: LDAP Control Extension for Server Side Sorting of Search Results provides server sorting of search results. The sort is performed based upon one or more attributes contained in the search results.

The **ldap_parse_sort_control()** routine can be used to extract the sort result code and failing attribute name from the sort results response control returned by the LDAP server.

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes listed in the ldap.h include file.

The following are some common client errors:

LDAP_CONTROL_NOT_FOUND
The server controls do not contain the sort results control

LDAP_NO_MEMORY
Insufficient storage is available

LDAP_PARAM_ERROR
A parameter is not valid

ldap_pwdpolicy_err2string()

Purpose

Return a descriptive text message for an LDAP password policy control response error or warning code

Format

```
#include <ldap.h>

char * ldap_pwdpolicy_err2string(
    int                 error)
```

Parameters

Input

error
Specifies the error or warning code. This can be obtained by issuing the **ldap_parse_pwdpolicy_response()** routine which returns the error code in `controlerrp` and the warning code in `controlwarnp`.

Usage

The **ldap_pwdpolicy_err2string()** routine, given a password policy control response (`1.3.6.1.4.1.42.2.27.8.5.1`) error or warning code from the **ldap_parse_pwdpolicy_response()** routine, returns a null-terminated character string that provides a textual description of the password policy error or warning.

For a password policy warning code, the **ldap_pwdpolicy_err2string()** routine returns a text string containing a substitution variable. The substitution value is returned in the `controlresp` parameter after issuing the **ldap_parse_pwdpolicy_response()** routine. To create the full warning text, use `printf` providing the text string containing the substitution variable obtained from this routine and the substitution value. For more information, see "ldap_parse_pwdpolicy_response()" on page 140.

The application must not modify or free this text message. The message is in the local EBCDIC code page or UTF-8, as determined by the `LDAP_LIBASCII` compiler variable

Function return value

The function return value is the address of the text message and is never a `NULL` address. The returned message is `N/A` if the LDAP message catalog cannot be accessed, storage cannot be allocated, or the error code is not a recognized LDAP error code.

ldap_remove_control()

Purpose

Remove a control from a list of controls

Format

```
#include <ldap.h>

int ldap_remove_control(
    LDAPControl *           control,
    LDAPControl ***         control_list,
    int                     freeit)
```

Parameters

Input

control
Specifies the control to be removed from the list of controls.

freeit
Specify `TRUE(1)` to free the control. Otherwise, specify `FALSE(0)`.

Output

control_list
Specifies the address of the control list.

Usage

The **ldap_remove_control()** routine removes a control from a list of controls. The control and its contents are freed if the *freeit* parameter is nonzero. The control is not freed if it is not found in the list of controls. The control address list is freed when all the controls have been removed.

Function return value

The function return value is `LDAP_SUCCESS` if no error is detected. Otherwise, the return value is one of the LDAP error codes that are listed in the `ldap.h` include file.

The following are some common client errors:

LDAP_CONTROL_NOT_FOUND
The control is not found in the list of controls.

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_rename(), ldap_rename_s()

Purpose

Rename an entry in the LDAP directory

Format

```
#include <ldap.h>

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

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

Parameters

Input

ld Specifies the LDAP handle.

dn Specifies the distinguished name for the directory entry as a null-terminated character string in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. A zero-length name is not allowed for a rename request.

newrdn
Specifies the new relative distinguished name (RDN) for the directory entry as a null-terminated character string in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle.

newparent
Specifies the distinguished name of the new parent entry as a null-terminated character string in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. Specify a zero-length character string to indicate that the root is the new parent. Specify NULL for this parameter to indicate that the parent entry is not to be changed. The LDAP protocol version must be LDAP_VERSION3 to specify a non-NULL value for this parameter.

deleteoldrdn
Specify TRUE(1) if the attributes from the old RDN are to be removed from the entry. Specify FALSE(0) if the attributes are to be retained.

serverctrls
Specifies an array of server controls for the rename request. The end of the array is indicated by a NULL address. If NULL is specified for this parameter, the server controls specified by the LDAP_OPT_SERVER_CONTROLS option for the LDAP handle are used. If NULL is specified for this parameter and the

LDAP_OPT_SERVER_CONTROLS option has not been set for the LDAP handle, no server controls are used. To override the server controls for the LDAP handle so that no controls are used, specify a server controls array consisting of a NULL address. (Control values for this routine vary depending on whether you are specifying server or client controls. See "LDAP controls" on page 15 for details.)

clientctrls
Specifies an array of client controls for the rename request. The end of the array is indicated by a NULL address. If NULL is specified for this parameter, the client controls specified by the LDAP_OPT_CLIENT_CONTROLS option for the LDAP handle are used. If NULL is specified for this parameter and the LDAP_OPT_CLIENT_CONTROLS option has not been set for the LDAP handle, no client controls are used. To override the client controls for the LDAP handle so that no controls are used, specify a client controls array consisting of a NULL address. (Control values for this routine vary depending on whether you are specifying server or client controls. See "LDAP controls" on page 15 for details.)

Output

msgidp
Returns the message identifier assigned to the rename request message. This value can be used when calling the **ldap_result()** routine to wait for the rename result message.

Usage

The **ldap_rename()** routine sends the request to the LDAP server and returns control to the application. The application must then call the **ldap_result()** routine to obtain the result.

The **ldap_rename_s()** routine sends the request to the LDAP server and then waits for the completion of the request. The rename request is abandoned if the client is unable to wait for the response because of an error from the **ldap_result()** routine.

The requested directory entry is renamed. The entry might or might not have subordinate entries. If the entry is not a leaf entry, the entire subtree is renamed.

Client controls specified by the LDAP_OPT_CLIENT_CONTROLS and server controls specified by the LDAP_OPT_SERVER_CONTROLS options are used by the **ldap_rename()** and **ldap_rename_s()** routines unless overridden by the *serverctrls* and *clientctrls* parameters.

Function return value

The **ldap_rename()** routine returns LDAP_SUCCESS if the request is sent to the LDAP server. Otherwise, the return value is one of the error codes listed in the ldap.h include file. Errors reported by the LDAP server are not returned by the **ldap_rename()** routine. Instead, the application must call the **ldap_parse_result()** routine to obtain the result code from the result message returned by the **ldap_result()** routine.

The **ldap_rename_s()** routine returns LDAP_SUCCESS if the request is successful. Otherwise, the return value is one of the error codes listed in the ldap.h include file. The return value includes errors detected by the LDAP client and errors detected by the LDAP server.

The following are some common client errors:

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_NOT_SUPPORTED
The LDAP protocol version must be LDAP_VERSION3 to specify server or client controls or to specify a new parent entry.

LDAP_PARAM_ERROR
A parameter is not valid.

LDAP_SERVER_DOWN
Network connection failed.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION
A critical client control is either not recognized or not supported for a rename operation.

The following are some common server errors:

LDAP_ALREADY_EXISTS
An entry with the new name exists.

LDAP_INSUFFICIENT_ACCESS
Not authorized to rename the directory entry.

LDAP_NO_SUCH_OBJECT
The directory entry does not exist.

LDAP_REFERRAL
The entry is not in the current LDAP server.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION
A critical server control is either not recognized or not supported for a rename operation.

ldap_result()

Purpose

Return the result message for an LDAP request

Format

```
#include <ldap.h>

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

Parameters

Input

ld Specifies the LDAP handle.

msgid
Specifies the message identifier assigned to the LDAP request. Specify LDAP_RES_ANY to return the next result message for the LDAP handle.

all
Specify LDAP_MSG_ONE to return a single search entry for a search request. Specify LDAP_MSG_ALL to return all the search entries and the search result for a search request.

timeout
Specifies the length of time to wait for a result message. This value is specified in the timeval structure that is defined in the time.h file. The timeval structure contains tv_sec and tv_usec fields for specifying the time in seconds and microseconds. Specify NULL for this parameter to wait until a result message is received. Set the tv_sec and tv_usec values in the timeout value to 0 to return immediately, if there is no result message available.

Output

result
Returns the result message. The application should call the **ldap_msgfree()** routine to release the message when it is no longer needed.

Usage

The **ldap_result()** routine returns the next result message for the LDAP handle. If there is no result message available and the timeout value is not 0, it waits for a message to be received.

The **ldap_result()** routine can be used with the asynchronous LDAP request routines to process result messages as they are returned by the LDAP server. The order of the result messages is not necessarily the same as the order in which the requests were sent to the LDAP server.

Function return value

The function return value is `0` if no result message is received before the timeout interval expires. The function return value is `-1` if an error occurs while receiving the result message, in which case the application can call the **ldap_get_errno()** routine to obtain the error code. Otherwise, the function return value is the message type for the first result message.

A search request can return multiple messages. There is a search entry message for each directory object satisfying the search criteria, plus a search result message after all the search entries are returned. There can also be search continuation reference messages if the LDAP client is not following referrals. By specifying `LDAP_MSG_ALL`, the application can make a single call to **ldap_result()** and get all the search messages at once (the **ldap_result()** routine does not return until the search result message is received). In this case, the function return value is `LDAP_RES_SEARCH_ENTRY` or `LDAP_RES_SEARCH_REFERENCE` if there is at least one search entry or search reference, and `LDAP_RES_SEARCH_RESULT` if there are no search entries or search references.

Errors detected while following referrals are returned in the result code for the result message and not as the function return value for the **ldap_result()** routine. Error messages returned by the LDAP server while following referrals are appended to the error string in the result message along with the referral value resulting in the error.

The following result message types can be returned:

LDAP_RES_ADD
: Add result.

LDAP_RES_BIND
: Bind result.

LDAP_RES_COMPARE
: Compare result.

LDAP_RES_DELETE
: Delete result.

LDAP_RES_EXTENDED
: Extended result.

LDAP_RES_MODIFY
: Modify result.

LDAP_RES_MODRDN
: Modify RDN result.

LDAP_RES_SEARCH_ENTRY
: Search entry.

LDAP_RES_SEARCH_REFERENCE
: Search reference.

LDAP_RES_SEARCH_RESULT
: Search result.

The following are some common errors for this routine:

LDAP_INVALID_STATE
A connection has not been established with the LDAP server or an unbind has been issued for the LDAP handle.

LDAP_LOCAL_ERROR
A local system error is detected.

LDAP_NO_MATCHING_REQUEST
The message identifier does not refer to an outstanding request.

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_PARAM_ERROR
A parameter is not valid.

LDAP_PROTOCOL_ERROR
Response message is not valid.

LDAP_SERVER_DOWN
A network error has occurred or the LDAP server has closed the connection.

LDAP_TIMEOUT
A response is not received within the timeout interval.

LDAP_WAIT_INTERRUPTED
A signal is received and the LDAP_OPT_RESTART option is not set to LDAP_OPT_ON.

ldap_sasl_bind(), ldap_sasl_bind_s()

Purpose

Bind to the LDAP server using the Simple Authentication and Security Layer (SASL)

Format

```
#include <ldap.h>

int ldap_sasl_bind(
    LDAP *                ld,
    const char *          who,
    const char *          mechanism,
    BerVal *              credentials,
    LDAPControl *         serverctrls[],
    LDAPControl *         clientctrls[],
    int *                 msgidp)

int ldap_sasl_bind_s(
    LDAP *                ld,
    const char *          who,
    const char *          mechanism,
    BerVal *              credentials,
    LDAPControl *         serverctrls[],
    LDAPControl *         clientctrls[],
    BerVal **             servercredp)
```

Parameters

Input

ld Specifies the LDAP handle.

who
Specifies the authorization name as a null-terminated character string. The name is in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. Specify NULL for this parameter if there is no authorization name.

mechanism
Specifies the security mechanism as a null-terminated character string. The mechanism is in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. The supported security mechanisms are NULL, EXTERNAL, GSSAPI, CRAM-MD5, and DIGEST-MD5. (These mechanisms are described in "Usage" on page 158.) A simple bind is performed if this parameter is NULL, the mechanism is a zero-length string, or the mechanism is NULL. The LDAP protocol version must be LDAP_VERSION3 for anything other than a simple bind.

credentials
Specifies the client credentials. Specify NULL for this parameter if there are no client credentials.

serverctrls
Specifies an array of server controls for the bind operation. The end of the array is indicated by a NULL address. Specify NULL for this parameter if there are no server controls. The bind operation then uses the server controls set by the LDAP_OPT_SERVER_CONTROLS option for the LDAP handle. If you do not want to use the server controls from the LDAP handle, specify the address of a server control list containing a single NULL address. (Control values for this

routine vary depending on whether you are specifying server or client controls. See "LDAP controls" on page 15 for details.)

clientctrls
Specifies an array of client controls for the bind operation. The end of the array is indicated by a NULL address. Specify NULL for this parameter if there are no client controls. The bind operation then uses the client controls set by the LDAP_OPT_CLIENT_CONTROLS option for the LDAP handle. If you do not want to use the client controls from the LDAP handle, specify the address of a client control list containing a single NULL address. (Control values for this routine vary depending on whether you are specifying server or client controls. See "LDAP controls" on page 15 for details.)

Output

msgidp
Returns the message identifier assigned to the bind request message. This value can be used when calling the **ldap_result()** routine to wait for the bind result message.

servercredp
Returns the server credentials from the result message for a synchronous bind request. The value is set to NULL if there are no server credentials. Specify NULL for this parameter if the server credentials are not needed. The application should call the **ldap_berfree_np()** routine to release the credentials when they are no longer needed.

Usage

The **ldap_sasl_bind()** or **ldap_sasl_bind_s()** routine binds to the LDAP server identified by the LDAP handle. The LDAP server authenticates the client using the specified Simple Authentication and Security Layer (SASL) mechanism.

The **ldap_sasl_bind()** routine sends the bind message to the LDAP server and returns control to the application. The application should call the **ldap_result()** routine to get the response to the bind request. The application can then call the **ldap_parse_result()** and **ldap_parse_sasl_bind_result()** routines to obtain information from the result message.

The **ldap_sasl_bind_s()** routine sends the bind message to the LDAP server and waits for a response. The bind request is abandoned if the client is unable to wait for the response because of an error from the **ldap_result()** routine.

The who parameter specifies the authorization name for the connection. This is also the authentication name for a simple bind. If the value of the who parameter is NULL, the LDAP server uses the authentication name from the SASL bind for authorization checking.

You can change the client authentication for a session by calling **ldap_sasl_bind()** or **ldap_sasl_bind_s()** again. Note some LDAP servers might not support changing the client authentication depending upon the SASL mechanism used to perform the initial authentication.

NULL mechanism

Using the NULL mechanism is equivalent to calling the **ldap_simple_bind()** or **ldap_simple_bind_s()** routine. For simple authentication, the client authenticates

itself to the server by supplying an authentication name and the password associated with the name. The authentication is successful if the password is correct. An anonymous bind is performed when no authentication name is supplied. LDAP supports simple authentication where the authentication name is the distinguished name of an entry in the LDAP directory. The password is the value associated with that directory entry.

Simple authentication is performed when the mechanism parameter is NULL, specifies a zero-length string, or specifies the string NULL. The who parameter specifies the distinguished name to be used as the authentication name. The bv_val and bv_len fields for the credentials parameter specify the password.

The ldap.h include file defines LDAP_SASL_SIMPLE and LDAP_MECHANISM_NULL for use as the mechanism value. The UTF-8 versions are LDAP_SASL_SIMPLE_UTF8 and LDAP_MECHANISM_NULL_UTF8.

Mutual authentication is not performed. The server verifies the identity of the client but the client has no way to verify the identity of the server.

Integrity and confidentiality services are not available and must be provided by the transport layer if they are needed (for example, by using SSL). Therefore, the LDAP_OPT_MIN_SASL_LEVEL and LDAP_OPT_MAX_SASL_LEVEL options are ignored for simple authentication and the LDAP_OPT_SASL_QOP option always returns a QOP of 0 (SASL provides no integrity or confidentiality services).

EXTERNAL mechanism - TCP/IP connection

For external authentication using a TCP/IP connection, the server authenticates the client using information external to the SASL protocol. LDAP supports external authentication using the X.509 client certificate provided by an SSL connection. The label specified on the call to the **ldap_ssl_init()** routine identifies the client certificate. System SSL selects a certificate if the application provides no label. See *z/OS Cryptographic Services System SSL Programming* for more information about how to specify a certificate using the GSK_KEY_LABEL environment variable or as the default certificate in a key database, SAF key ring, or PKCS #11 token.

External authentication is performed when the *mechanism* parameter is the string EXTERNAL. To use external authentication with LDAP, the client must use an SSL connection to the LDAP server and must not provide any credentials when calling the **ldap_sasl_bind()** or **ldap_sasl_bind_s()** routine (the credentials parameter must either be NULL or point to a zero-length value). The LDAP server uses the subject name from the client certificate as the authentication name.

The ldap.h include file defines LDAP_MECHANISM_EXTERNAL for use as the mechanism value. The UTF-8 version is LDAP_MECHANISM_EXTERNAL_UTF8.

SSL performs mutual authentication. The server verifies the identity of the client using the client certificate and the client verifies the identity of the server using the server certificate.

SSL provides integrity and confidentiality services.

System SSL must be installed to use the EXTERNAL SASL mechanism. The LDAP_OPT_MIN_SASL_LEVEL and LDAP_OPT_MAX_SASL_LEVEL options are ignored for external authentication and the LDAP_OPT_SASL_QOP option always returns a QOP of

0 (SASL provides no integrity or confidentiality services). You can use the LDAP_OPT_SSL_CIPHER option to obtain the SSL cipher suite negotiated by the LDAP client and the LDAP server.

GSSAPI mechanism

For GSSAPI authentication, the server authenticates the client using Kerberos Version 5 credentials. The client is responsible for obtaining a Kerberos ticket-granting ticket (TGT) for the wanted client identity. The user can obtain the TGT by using the **kinit** command or the application can obtain the TGT by calling the appropriate Kerberos API routines. For more information about the **kinit** command, see *z/OS Integrated Security Services Network Authentication Service Administration*. For more information about the Kerberos and GSSAPI routines, see *z/OS Integrated Security Services Network Authentication Service Programming*.

The system where the LDAP server is running must be defined to the DNS name server. If an IP address is supplied on the call to **ldap_init()** or **ldap_ssl_init()**, the name server must be able to translate the IP address to a host name. The LDAP server must have a Kerberos principal name in one of the following forms:

LDAP/*primary-host-name***@***realm-name*

or

ldap/*primary-host-name***@***realm-name*

The LDAP client first tries to obtain a Kerberos service ticket to **LDAP/***primary-host-name***@***realm-name* and retries using **ldap/***primary-host-name***@***realm-name* if the server principal is not defined. The primary host name is the canonical name returned by the DNS name server and consists of lowercase characters. (Although DNS name are not case-sensitive, Kerberos principal names are case-sensitive).

GSSAPI authentication is performed when the mechanism parameter is the string GSSAPI. The application can either acquire the GSSAPI credential before calling the **ldap_sasl_bind()** or **ldap_sasl_bind_s()** routine or it can use the default GSSAPI credential. If the application acquires the GSSAPI credential, the value of the credentials parameter must be the credential identifier (the bv_val field contains the address of the credential identifier and the bv_len field is the length of the credential identifier). The LDAP client uses the TGT from the Kerberos credentials cache to acquire a GSSAPI credential if the credentials parameter is NULL or points to a zero-length value. The Kerberos mechanism can be used to perform the authentication with the LDAP server. The authentication name is the Kerberos client principal obtained from the TGT. Delegated credentials are made available to the LDAP server if the LDAP_OPT_DELEGATION option is set to LDAP_OPT_ON.

The ldap.h include file defines LDAP_MECHANISM_GSSAPI for use as the mechanism value. The UTF-8 version is LDAP_MECHANISM_GSSAPI_UTF8.

Mutual authentication is performed by GSSAPI. The server verifies the identity of the client when the client demonstrates that it knows the session key contained in the encrypted service ticket. The client verifies the identity of the server when the server demonstrates that it knows the encryption key for the service ticket.

Integrity and confidentiality services are available when offered by the LDAP server. The LDAP_OPT_MIN_SASL_LEVEL option sets the minimum protection level and defaults to LDAP_SASL_LEVEL_NONE. The bind fails if the LDAP server does not offer

at least this level of protection. The LDAP_OPT_MAX_SASL_LEVEL option sets the maximum protection level and defaults to LDAP_SASL_LEVEL_CONF. The LDAP client does not negotiate a higher protection level even if the server offers it. The LDAP_OPT_SASL_QOP option can be used to obtain the negotiated integrity and protection levels.

Network Authentication Services must be installed to use the GSSAPI SASL mechanism.

CRAM-MD5 mechanism

For CRAM-MD5 authentication, the client authenticates itself to the server by supplying an authentication name and the password associated with the name. Unlike simple authentication where the password is sent to the LDAP server, CRAM-MD5 uses a challenge-response message exchange which never sends the password to the server. Instead, the password can be used as the shared secret to generate a keyed MD5 digest. The client sends this digest to the server, which generates its own digest. If the server digest matches the client digest, the client is authenticated by the server.

CRAM-MD5 authentication is performed when the mechanism parameter is the string CRAM-MD5. The authentication name is the short name specified by the ibm-saslBindCramUserName client control or the DN of a directory object if specified by the who parameter (the short name can be used if both are specified). An error is returned if the short name contains any blanks or the ibm-saslBindCramUserName client control is specified more than once. A parameter error is returned if the DN or the ibm-saslBindCramUserName client control is specified. The bv_val and bv_len fields for the *credentials* parameter specify the password. A parameter error is returned if no password is provided. All strings are in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. In addition, the short name and password must consist of characters that can be represented in the ISO8859-1 code page.

The ldap.h include file defines LDAP_MECHANISM_CRAM for use as the mechanism value. The UTF-8 version is LDAP_MECHANISM_CRAM_UTF8.

Mutual authentication is not performed. The server verifies the identity of the client but the client has no way to verify the identity of the server.

Integrity and confidentiality services are not available and must be provided by the transport layer if they are needed (for example, by using SSL). Therefore, the LDAP_OPT_MIN_SASL_LEVEL and LDAP_OPT_MAX_SASL_LEVEL options are ignored for CRAM-MD5 authentication and the LDAP_OPT_SASL_QOP option always returns a QOP of 0 (no integrity or confidentiality services are provided by SASL).

DIGEST-MD5 mechanism

For DIGEST-MD5 authentication, the client authenticates itself to the server by supplying an authentication name and the password associated with the name. Unlike simple authentication, where the password is sent to the LDAP server, DIGEST-MD5 uses a challenge-response message exchange which never sends the password to the server. Instead, the password can be used as the shared secret to generate a keyed MD5 digest. The client sends this digest to the server, which generates its own digest. If the server digest matches the client digest, the client is authenticated by the server. The server then generates a response digest and sends it to the client, which generates its own digest. If the client digest matches the

server digest, the server is authenticated by the client. Therefore, the DIGEST-MD5 mechanism provides mutual authentication while the CRAM-MD5 mechanism provides just client authentication.

DIGEST-MD5 authentication is performed when the mechanism parameter is the string DIGEST-MD5. The ibm-saslBindDigestRealmName client control specifies the digest realm. The digest realm can be used to select the authentication realm when the LDAP server supports multiple realms. If this control is not specified and the LDAP server does not specify a realm, the local host name can be used as the digest realm. A parameter error is returned if the ibm-saslBindDigestRealmName client control is specified more than once. The ibm-saslBindDigestUserName client control specifies the user name. A parameter error is returned if the ibm-saslBindDigestUserName client control is not specified or is specified more than once. The bv_val and bv_len fields for the credentials parameter specify the password. If no password is provided, a parameter error is returned. The who parameter provides an optional authorization distinguished name that is sent to the LDAP server as part of the DIGEST-MD5 message exchange. All strings are in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle.

The ldap.h include file defines LDAP_MECHANISM_DIGEST for use as the mechanism value. The UTF-8 version is LDAP_MECHANISM_DIGEST_UTF8.

The DIGEST-MD5 mechanism performs mutual authentication. The server verifies the identity of the client when the client demonstrates that it knows the password by sending the correct request digest. The client verifies the identity of the server when the server demonstrates that it knows the password by sending the correct response digest.

Integrity and confidentiality services are available when offered by the LDAP server. The LDAP_OPT_MIN_SASL_LEVEL option sets the minimum protection level and defaults to LDAP_SASL_LEVEL_NONE. The bind fails if the LDAP server does not offer at least this level of protection. The LDAP_OPT_MAX_SASL_LEVEL option sets the maximum protection level and defaults to LDAP_SASL_LEVEL_CONF. The LDAP client does not negotiate a higher protection level even if the server offers it. The LDAP_OPT_SASL_QOP option can be used to obtain the negotiated integrity and protection levels.

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes listed in the ldap.h include file. The **ldap_sasl_bind_s()** routine returns errors reported by the LDAP server and errors detected by the LDAP client. The **ldap_sasl_bind()** routine does not return errors reported by the LDAP server. Instead, the application must call the **ldap_parse_result()** routine to obtain the result code from the bind response message returned by the **ldap_result()** routine. Errors detected by the LDAP client run time during the SASL negotiation are also returned in the bind result message.

The following are some common client errors:

LDAP_GSS_INIT_FAILED
: Kerberos GSS-API initialization failed.

LDAP_GSS_NOT_AVAILABLE
: Kerberos GSS-API support is not available.

LDAP_INVALID_STATE
A bind or unbind is in progress for the LDAP handle or an application exit is active for the LDAP handle.

LDAP_LOCAL_ERROR
A system function detected an error.

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_NOT_SUPPORTED
The LDAP protocol version is not version 3 and the mechanism is not simple or external authentication.

LDAP_PARAM_ERROR
A parameter is not valid.

LDAP_SASL_INAPPROPRIATE
The LDAP server does not offer a security level that meets the criteria set by the application.

LDAP_SERVER_DOWN
Unable to connect to the LDAP server.

LDAP_SSL_NOT_USED
The EXTERNAL mechanism is requested but the connection is not using SSL.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION
A critical client control is either not recognized or is not supported for a bind operation.

The following are some common bind result codes:

LDAP_DIGEST_MD5_SASL_FAILED
SASL DIGEST-MD5 negotiation failed.

LDAP_GSS_SASL_FAILED
SASL GSS-API negotiation failed.

LDAP_INAPPROPRIATE_AUTH
Inappropriate authentication provided by the client.

LDAP_INVALID_CREDENTIALS
The credentials provided by the client are not valid.

LDAP_PROTOCOL_ERROR
A protocol error is detected during the SASL negotiation.

LDAP_REFERRAL
The server cannot accept the bind.

LDAP_STRONG_AUTH_NOT_SUPPORTED
The server does not support the requested SASL mechanism.

LDAP_STRONG_AUTH_REQUIRED
The server requires strong authentication.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION
A critical server control is either not recognized or is not supported for a bind operation.

ldap_search(), ldap_search_s(), ldap_search_st(), ldap_search_ext(), ldap_search_ext_s()

Purpose

Search the LDAP directory

Format

```
#include <ldap.h>

int ldap_search(
    LDAP *                ld,
    const char *          base,
    int                   scope,
    const char *          filter,
    const char *          attrs[],
    int                   attrsonly)

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

int ldap_search_st(
    LDAP *                ld,
    const char *          base,
    int                   scope,
    const char *          filter,
    const char *          attrs[],
    int                   attrsonly,
    struct timeval *      timeout,
    LDAPMessage **        result)

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

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

Parameters

Input

ld Specifies the LDAP handle.

base
Specifies the distinguished name of the directory object where the search should start. The name is a null-terminated character string in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. The distinguished name should be in the format that is defined by RFC 2253: *UTF-8 String Representation of Distinguished Names*.

scope
Specifies the search scope as follows:

LDAP_SCOPE_BASE
Search just the entry that is specified by the base name.

LDAP_SCOPE_ONELEVEL
Search the immediate children of the base entry.

LDAP_SCOPE_SUBTREE
Search the base entry and all of its descendants.

filter
Specifies the search filter as a null-terminated character string in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. If you specify NULL or a zero-length string for this parameter, the search filter is set to "(objectClass=*)". For information about filter syntax, see "Usage" on page 167.

attrs
Specifies an array of attribute types to be returned. Each attribute type is a null-terminated character string in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8-IO option for the LDAP handle. The end of the array is indicated by a NULL address. If you specify NULL for this parameter, all the attributes for an entry are returned.

attrsonly
Specifies whether the attribute values should be returned along with the attribute types. A nonzero value causes just the attribute types to be returned. A zero value causes both attribute types and attribute values to be returned.

serverctrls
Specifies an array of server controls for the search request. The end of the array is indicated by a NULL address. If NULL is specified for this parameter, the server controls specified by the LDAP_OPT_SERVER_CONTROLS option for the LDAP handle are used. If NULL is specified for this parameter and the LDAP_OPT_SERVER_CONTROLS option has not been set for the LDAP handle, no server controls are used. To override the server controls for the LDAP handle so that no controls are used, specify a server controls array consisting of a NULL address. (Control values for this routine vary depending on whether you are specifying server or client controls. See "LDAP controls" on page 15 for details.)

clientctrls
Specifies an array of client controls for the search request. The end of the array is indicated by a NULL address. If NULL is specified for this parameter, the client controls specified by the LDAP_OPT_CLIENT_CONTROLS option for the LDAP handle are used. If NULL is specified for this parameter and the

LDAP_OPT_CLIENT_CONTROLS option has not been set for the LDAP handle, no client controls are used. To override the client controls for the LDAP handle so that no controls are used, specify a client controls array consisting of a NULL address. (Control values for this routine vary depending on whether you are specifying server or client controls. See "LDAP controls" on page 15 for details.)

timeout
Specifies the maximum time for the search request. This value is specified in the timeval structure that is defined in the time.h file. The timeval structure contains tv_sec and tv_usec fields for specifying the time in seconds and microseconds. Specify NULL for this parameter if there is no time limit for the request. Otherwise, set the tv_sec and tv_usec values in the *timeout* value to the maximum time in seconds and microseconds. For the **ldap_search_ext()** routine, the value of the tv_sec value in the *timeout* parameter is sent to the server and overrides the value of LDAP_OPT_TIMELIMIT in the LDAP handle. For the **ldap_search_ext_s()** routine, the value of the tv_sec value in the *timeout* parameter is sent to the server (overriding the value of LDAP_OPT_TIMELIMIT in the LDAP handle) and specifies how long the client waits before abandoning the request. The tv_usec values in the *timeout* parameter are ignored in the **ldap_search_ext()** and **ldap_search_ext_s()** routines. For the **ldap_search_st()** routine, the value of LDAP_OPT_TIMELIMIT in the LDAP handle is sent to the server to indicate a limit on the search time in the server while the values of tv_sec and tv_usec in the *timeout* parameter specifies how long the client waits before abandoning the request.

The LDAP server can also provide a limit on the search time. For information about the server's search time limit and how it interacts with the client time limit, see the documentation for your LDAP server. For the IBM Tivoli Directory Server for z/OS, see the description of the **timeLimit** configuration file option (Customizing the LDAP server configuration) in *z/OS IBM Tivoli Directory Server Administration and Use for z/OS*. The default time limit for the client, which is specified by a value of 0, indicates that there is no client time limit and that the maximum number of seconds is limited only by the LDAP server limit.

sizelimit
Specifies the maximum number of entries that can be returned, overriding the value of LDAP_OPT_SIZELIMIT in the LDAP handle. A value of 0 indicates that there is no limit.

The LDAP server can also provide a size limit on the number of entries returned. For information about the server's size limit and how it interacts with the client size limit, see the documentation for your LDAP server. For the IBM Tivoli Directory Server for z/OS, see the description of the **sizeLimit** configuration file option (Customizing the LDAP server configuration) in *z/OS IBM Tivoli Directory Server Administration and Use for z/OS*. The default size limit for the client, which is specified by a value of 0, indicates that the maximum number of entries is limited only by the LDAP server limit.

Output

result
Returns the address of the result message chain. The message address is set to NULL if there are no result messages returned by the LDAP server. Note that the synchronous routines can return one or more result messages even when the

function return value is not LDAP_SUCCESS. The application should call the **ldap_msgfree()** routine to release the message chain when it is no longer needed.

msgidp
Returns the message identifier that is assigned to the search request message. This value can be used when calling the **ldap_result()** routine to wait for the search results.

Usage

The **ldap_search()** and **ldap_search_ext()** routines initiate the search and return control to the application. The application must then call the **ldap_result()** routine to obtain the search results.

The **ldap_search_s()**, **ldap_search_st()** and **ldap_search_ext_s()** routines initiate the search and wait for the search results. The **ldap_search_s()** routine waits indefinitely, and the **ldap_search_st()** and **ldap_search_ext_s()** routines provide a parameter to specify a time limit. The search request is abandoned if the client is unable to wait for the response because of an error from the **ldap_result()** routine. The search request is also abandoned if the time limit specified for the **ldap_search_st()** routine expires.

The **ldap_search_ext()** routine uses the *timeout* parameter to specify the maximum time for the search request. A search issued by **ldap_search_ext()** is terminated by the LDAP server with a result code of LDAP_TIMELIMIT_EXCEEDED when the time limit is exceeded. The **ldap_search_st()** routine uses the *timeout* parameter to specify how long the client should wait for a response. A search issued by **ldap_search_st()** is abandoned by the LDAP client with a result code of LDAP_TIMEOUT when the time limit is exceeded. The **ldap_search_ext_s()** routine uses the timeout parameter to specify the maximum time for the search request and how long the client should wait for a response. A search issued by **ldap_search_ext_s()** might either be terminated by the LDAP server with a result code of LDAP_TIMELIMIT_EXCEEDED or by the LDAP client with a result code of LDAP_TIMEOUT when the time limit is exceeded. The result code that is returned from the **ldap_search_ext_s()** routine depends on where the time limit is first triggered. For example, a client timeout can occur if the server has sent all requested search entries but a network delay occurs that prevents the client from receiving all requested search entries. A server-side timeout can occur if the timeout value that is specified is not long enough for the server to retrieve all the requested entries on the search request. Setting the *tv_sec* field to 0 for **ldap_search_ext()** indicates that there is no time limit for the search request, while setting the *tv_sec* field to 0 for **ldap_search_st()** or **ldap_search_ext_s()** indicates that the client should not wait for a response (which means that the search request is abandoned if the response is not immediately available).

The LDAP server returns zero or more search entry and search continuation reference messages, followed by the search done message. There is a search entry message for each directory object that matched the search criteria. There are search continuation reference messages if the LDAP server is unable to search all objects in the scope under the base object. The search done message indicates any errors encountered during the search.

Search continuation references are handled by the LDAP client run time if the LDAP_OPT_REFERRALS option is set for the LDAP handle. (This is the default). In this case, the application does not receive the search continuation reference messages,

because they are replaced by search entry messages obtained by following the referral. Errors encountered while following referrals are added to the error text in the search done message. If multiple errors are detected, the error text contains a line for each error and the result code indicates the first error.

Use the **ldap_first_entry()** and **ldap_next_entry()** routines to process the result message chain. If referrals are not handled by the LDAP client, use the **ldap_first_reference()** and **ldap_next_reference()** routines to handle any search continuation references. Call the **ldap_msgfree()** routine to release the result message chain after all the messages have been processed.

Server and client controls specified by the LDAP_OPT_SERVER_CONTROLS and LDAP_OPT_CLIENT_CONTROLS options are used by all the search routines but are overridden by the *timeout* and *sizelimit* values for the **ldap_search_ext()** and **ldap_search_ext_s()** routines. The search time limit specified by the LDAP_OPT_TIMELIMIT option and the search size limit specified by the LDAP_OPT_SIZELIMIT option are used by the **ldap_search()** and **ldap_search_s()** routines and can be overridden for the **ldap_search_ext()** and **ldap_search_ext_s()** routines. For the **ldap_search_st()** routine, the LDAP_OPT_TIMELIMIT value is sent to the server in the search request and the *timeout* value can be used to determine how long to wait for the server response.

The search results can be cached if a search result cache is specified for the LDAP handle. These results can be used to satisfy subsequent search requests without sending the search request to the LDAP server. You can use the ibm-serverHandledSearchRequest client control to disable caching for a specific search request. For more information about search caching, see "Client-side search results caching" on page 20.

Constructing search filters

Search filters are constructed as defined in RFC 2254: *String Representation of LDAP Search Filters*. The filter syntax is defined by the rules that are shown below.

Rules:

```
filter        =  "(" filtercomp ")"
filtercomp    =  and / or / not / item
and           =  "&" filterlist
or            =  "|" filterlist
not           =  "!" filter
filterlist    =  1*filter
item          =  simple / present / substring / extensible
simple        =  attr filtertype value
filtertype    =  equal / approx / greatereq / lesseq
equal         =  "="
approx        =  "~="
greatereq     =  ">="
lesseq        =  "<="
extensible    =  attr [":dn"] [":" matchingrule] ":=" value
                    / [":dn"]  ":" matchingrule  ":=" value
present       =  attr "=*"
substring     =  attr "=" [initial] any [final]
initial       =  value
any           =  "*" *(value "*")
final         =  value
attr          =  AttributeDescription as defined in RFC 2251
matchingrule  =  MatchingRuleId as defined in RFC 2251
value         =  AttributeValue as defined in RFC 2251
```

Values inside double quotation marks represent literal values. Items that are enclosed in square brackets are optional. Items that are separated by / represent a choice. The notation `1*filter` indicates one or more filters. Specifying `":dn"` as part of the extensible item indicates that the components of the distinguished name are to be included in the matching and the object attributes.

An error is returned if an extensible filter item is specified and the LDAP protocol version is not `LDAP_VERSION3`.

Note: The z/OS LDAP server does not support extensible search filters. Approximate search filters are treated as equality search filters in the z/OS LDAP server.

Leading and trailing white space characters are ignored. Embedded white space characters are allowed within an attribute value and are retained. Embedded white space characters are not allowed within any of the literals in the above rules. Quotation marks have no special meaning within a search filter and are treated as normal characters.

Filter control characters, such as `"("`, `")"`, `"*"`, and `"\"`, within an attribute value must be escaped using the format `"\`*xx*`"`, where *xx* is the hexadecimal representation of the ASCII value of the escaped character. For example, `"*"` would be represented as `"\2a"`. UTF-8 characters can be represented as a sequence of escaped characters; for example, `"(sn=Lu\c4\8di\c4\87)"`. The case of the hexadecimal characters is not important.

IETF RFC 2254 replaces IETF RFC 1960. However, RFC 1960 specified that filter control characters were escaped by preceding the escaped control character with a reverse slash. For example, `"*"` would be represented as `"\*"` within an attribute value. To provide compatibility with applications written to RFC 1960, filter control characters can be escaped using either format.

Earlier levels of LDAP allowed the outer parentheses to be omitted from the filter. For compatibility, z/OS LDAP allows the outer parentheses to be omitted from the filter. For example, `"mail=*"` can be specified instead of `"(mail=*)"`.

Examples: The following are some examples of filters:

- `(mail=*)`

  This filter matches any entry with the `mail` attribute and does not match entries without the `mail` attribute.
- `(mail=*@student.of.life.edu)`

  This filter matches any entry whose `mail` attribute value ends with the string `"@student.of.life.edu"`.
- `(&(cn=Jane*)(sn=Doe)(!(uid=jdoe)))`

  This filter matches any entry whose `cn` attribute value starts with `Jane` and whose `sn` attribute value is `Doe` and whose `uid` attribute value is not `jdoe`.

Function return value

The **ldap_search()** routine returns `-1` if a client error is detected. Otherwise, it returns the message identifier that is assigned to the search request. If the return value is `-1`, the application should call the **ldap_get_errno()** routine to get the error code. Errors reported by the LDAP server are not returned by the **ldap_search()**

routine. Instead, the application must call the **ldap_parse_result()** routine to obtain the result code from the search done message returned by the **ldap_result()** routine.

The **ldap_search_ext()** routine returns LDAP_SUCCESS if the search request is sent to the LDAP server. Otherwise, the return value is one of the error codes listed in the ldap.h include file. The **ldap_search_ext()** routine does not return errors reported by the LDAP server. The application must call the **ldap_parse_result()** routine to obtain the result code from the search done message returned by the **ldap_result()** routine.

The **ldap_search_s()**, **ldap_search_st()** and **ldap_search_ext_s()** routines return LDAP_SUCCESS if the request is successful. Otherwise, the return value is one of the error codes listed in the ldap.h include file. The return value includes errors detected by the LDAP client and errors detected by the LDAP server. One or more result messages can be returned by these routines even when the return value is not LDAP_SUCCESS. If no result messages are returned, the result message address is NULL.

The following are some common client errors:

LDAP_FILTER_ERROR
The search filter is not valid.

LDAP_INVALID_STATE
An unbind request has been issued for the LDAP handle.

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_NOT_SUPPORTED
The LDAP protocol version must be LDAP_VERSION3 to use an extensible filter item or to specify server or client controls.

LDAP_PARAM_ERROR
A parameter is not valid.

LDAP_SERVER_DOWN
Network connection failed.

LDAP_TIMEOUT
The wait time has expired and the search request has been abandoned.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION
A critical client control is either not recognized or is not supported for a search operation.

The following are some common search result codes:

LDAP_INSUFFICIENT_ACCESS
Not authorized to access base object.

LDAP_NO_SUCH_OBJECT
The base object is not found.

LDAP_REFERRAL
The base object is not in the current LDAP server.

LDAP_SIZELIMIT_EXCEEDED
The search size limit has been exceeded.

LDAP_TIMELIMIT_EXCEEDED
The search time limit has been exceeded.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION
A critical server control is either not recognized or is not supported for a search operation.

ldap_server_conf_save()

Purpose

Save the LDAP server information list

Format

```
#include <ldap.h>

int ldap_server_conf_save(
    const char *                filename,
    unsigned long               ttl,
    LDAPServerInfo *            server_info_list)
```

Parameters

Input

filename
Specifies the name of the server information file as a null-terminated string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable. Specify NULL to use the default server information file /etc/ldap/ldap_server_info.conf.

ttl
Specifies the time-to-live period in minutes for the information in the server information file. Specify 0 if the server information file has no expiration time and it remains valid until it is rewritten. After the *ttl* period expires, the information in the server information file is ignored.

server_info_list
Specifies the server information list. For a description of the server information list, see "ldap_server_locate()" on page 175. Text data is in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable.

Usage

The **ldap_server_conf_save()** routine saves the LDAP server information list returned by the **ldap_server_locate()** routine. The *filename* parameter specifies the file in which the server information is saved. This file is rewritten each time the **ldap_server_conf_save()** routine should be called. (For details, see "LDAP server information file" on page 246.) An error is returned if the directory path does not exist.

Access information from the server information list saved in the server information file can be used on subsequent calls to the **ldap_server_locate()** routine to eliminate the need to contact the DNS name server for the information.

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes listed in the ldap.h include file.

The following are some common errors for this routine:

LDAP_INSUFFICIENT_ACCESS
Not authorized to update the server information file.

LDAP_LOCAL_ERROR
An error occurred while writing the server information file.

ldap_server_free_list()

Purpose

Release a server information list

Format

```
#include <ldap.h>

int ldap_server_free_list(
    LDAPServerInfo *      server_info_list)
```

Parameters

Input

server_info_list
Specifies the first entry in the list.

Usage

The **ldap_server_free_list()** routine releases the storage allocated for a server information list returned by the **ldap_server_locate()** routine. All entries are released starting with the entry specified by the *server_info_list* parameter.

Function return value

The function return value is always LDAP_SUCCESS.

ldap_server_locate()

Purpose

Locate the LDAP servers

Format

```
#include <ldap.h>

int ldap_server_locate(
    LDAPServerRequest *        server_request,
    LDAPServerInfo **          server_info_list)
```

Parameters

Input

server_request
Specifies the address of an LDAPServerRequest structure. The application should initialize the structure to 0 before setting specific fields in the structure, to ensure that defaults are used when a field is not explicitly set. If you want the default behavior for all fields, specify NULL for this parameter. Text data is in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable.

Output

server_info_list
Returns the address of the first LDAPServerInfo structure in a list of LDAPServerInfo structures. Each LDAPServerInfo structure contains the address of the next structure in the list. The end of the list is indicated by a NULL address. The application should call the **ldap_server_free_list()** routine to release the list when it is no longer needed. Text data is in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable.

Usage

Use the **ldap_server_locate()** routine to locate one or more LDAP servers. Specify NULL for the *server_request* parameter to use the default request values. (For details about information contained in the server information file, see "LDAP server information file" on page 246.)

The LDAPServerRequest structure is defined as follows:

```
typedef struct LDAP_Server_Request {
    int                search_source;
    char *             conf_filename;
    int                reserved;
    char *             service_key;
    char *             enetwork_domain;
    char **            name_servers;
    char **            dns_domains;
    int                connection_type;
    int                connection_timeout;
    char *             DN_filter;
    char *             proto_key;
    unsigned char      reserved2[60];
} LDAPServerRequest;
```

where:

search_source
Specifies the search order as follows:

LDAP_LSI_CONF_DNS
Causes the server information file to be searched followed by DNS if no matching entries are found in the server information file or if the server information file has expired. (This is the default.)

LDAP_LSI_CONF_ONLY
Causes only the server information file to be searched.

LDAP_LSI_DNS_ONLY
Causes only DNS to be searched.

conf_filename
Specifies the server information file name. Specify `NULL` to use the default server information file `/etc/ldap/ldap_server_info.conf`. Otherwise, specify the address of a null-terminated string. This field is ignored if `LDAP_LSI_DNS_ONLY` is specified.

service_key
Specifies the service key used to form the DNS resource name. Specify `NULL` to use the default service key of `ldap`. Otherwise, specify the address of a null-terminated string consisting of characters that can be represented in the ISO8859-1 code page and having a maximum length of 63 characters. If `NULL` is specified for *service_key* and the search is unsuccessful, the **ldap_server_locate()** routine retries the search using `_ldap` for the service key (and `_tcp` for the protocol key if `NULL` is also specified for the *proto_key* field). Note `_ldap` is the preferred service key as defined by the latest version of RFC 2052: *A DNS RR for specifying the location of services (DNS SRV)*. The application should specify a service key of `_ldap` to bypass the double search if the `ldap` service key is not being used.

enetwork_domain
Specifies the eNetwork domain name used to form the DNS resource name. Specify `NULL` to use the default eNetwork domain name obtained from the `ldap_user_info` configuration file in the home directory for the current user. (This file is created by the **ldap_enetwork_domain_set()** routine.) Otherwise, specify the address of a null-terminated string consisting of characters that can be represented in the ISO8859-1 code page and having a maximum length of 63 characters. No eNetwork domain name can be used if `NULL` is specified for the *enetwork_domain* field and the **ldap_enetwork_domain_set()** routine has not been called to set a default eNetwork domain name for the user. The application can override the default eNetwork domain and use no eNetwork domain by specifying a zero-length string as the *enetwork_domain* value.

An eNetwork domain is a naming construct, implemented by the LDAP administrator, to further subdivide a set of LDAP servers (as published in DNS) into logical groupings. When you specify an eNetwork domain, only the LDAP servers grouped within the specified eNetwork domain are returned. This is useful when an application, or group of applications, needs access to a particular set of LDAP servers within the enterprise. For example, the research division within a company might use a dedicated set of LDAP servers. By publishing this dedicated set of LDAP servers in DNS with an eNetwork domain of `research`, applications that need to access information published in the research division's LDAP servers can selectively obtain the host names and ports of just those servers. Other LDAP servers also published in DNS are not returned.

name_servers
Specifies a list of domain name servers. Specify NULL to use the default domain name servers. Otherwise, specify the address of an address array where each array entry is the address of a null-terminated character string consisting of characters that can be represented in the ISO8859-1 code page. The address array is terminated by a NULL entry. Each character string represents the IP address of a domain name server specified in dotted decimal (IPv4) or colon-hexadecimal (IPv6) format. The default domain name servers are obtained from the resolver configuration file specified by the RESOLVER_CONFIG environment variable. The default resolver configuration file /etc/resolv.conf can be used if the RESOLVER_CONFIG environment variable is not defined. For information about the contents of the name resolver configuration file, see "Name resolver configuration file" on page 244.

Each name server in the list is queried in the specified order until either a successful answer to the query is received or an authoritative answer is received indicating the resource name is not known.

dns_domains
Specifies a list of domain names used to form the DNS resource name. Specify NULL to use the default domain name list. Otherwise, specify the address of an address array where each array entry is the address of a null-terminated character string consisting of characters that can be represented in the ISO8859-1 code page. The address array is terminated by a NULL entry. Each character string represents a domain name, such as endicott.ibm.com. The default domain names are obtained from the resolver configuration file specified by the RESOLVER_CONFIG environment variable. The default resolver configuration file /etc/resolv.conf can be used if the RESOLVER_CONFIG environment variable is not defined. For information about the contents of the name resolver configuration file, see "Name resolver configuration file" on page 244.

A search is performed for each domain name in the domain name list. The server information list returned to the application contains the results of all the searches. The entries are ordered as specified in the domain name list. That is, all entries matching the first domain name are followed by all entries matching the second domain name; all entries matching the second domain name are followed by all entries matching the third domain name; and so on. The entries within each domain are ordered based on priority and weight as described in RFC 2052: *A DNS RR for specifying the location of services (DNS SRV)*.

connection_type
Specifies the type of connection used to communicate with the domain name server as follows:

LDAP_LSI_TCP
Causes only TCP to be used.

LDAP_LSI_UDP
Causes only UDP to be used.

LDAP_LSI_UDP_TCP
Causes UDP to be used followed by TCP if the name server answer is truncated. (This is the default.)

connection_timeout
Specifies the amount of time in seconds to wait for a response from the name server. Specify 0 to use the default timeout value. Otherwise, specify the number of seconds to wait for a response. The default timeout value is obtained from the resolver configuration file specified by the RESOLVER_CONFIG

environment variable. The default resolver configuration file /etc/resolv.conf can be used if the RESOLVER_CONFIG environment variable is not defined. The default timeout is 5 seconds if the resolver configuration file does not contain a timeout value. For information about the contents of the name resolver configuration file, see "Name resolver configuration file" on page 244.

DN_filter
Specifies the naming context you want. LDAP servers that do not provide a naming context which includes the specified distinguished name are not included in the server list. Specify NULL to include all LDAP servers. Otherwise, specify the address of a null-terminated string consisting of characters that can be represented in the ISO8859-1 code page. The server list is sorted so that the best matches are listed first. For example, if the filter DN is "cn=Mary, sn=Roberts, ou=Bose, o=Acme, c=US" and LDAP ServerA supports naming context "o=Acme,c=US" and LDAP ServerB supports naming context "ou=Bose,o=Acme,c=US", then ServerB is returned before ServerA.

proto_key
Specifies the protocol key used to form the DNS resource name. Specify NULL to use the default protocol key of tcp. Otherwise, specify the address of a null-terminated string consisting of characters that can be represented in the ISO8859-1 code page and having a maximum length of 63 characters. If NULL is specified for *proto_key* and the search is unsuccessful, the **ldap_server_locate()** routine retries the search using _tcp for the protocol key (and _ldap for the service key if NULL is also specified for the *service_key* field). Note _tcp is the preferred protocol key as defined by the latest version of RFC 2052: *A DNS RR for specifying the location of services (DNS SRV)*. The application should specify a protocol key of _tcp to bypass the double search if the tcp protocol key is not being used. The protocol key is ignored when looking for an entry in the server information file.

The LDAPServerInfo structure is defined as follows:

```
typedef struct LDAP_Server_Info {
    char *                  lsi_host;
    unsigned short          lsi_port;
    char *                  lsi_suffix;
    char *                  lsi_query_key;
    char *                  lsi_dns_domain;
    int                     lsi_replica_type;
    int                     lsi_sec_type;
    unsigned short          lsi_priority;
    unsigned short          lsi_weight;
    char *                  lsi_vendor_info;
    char *                  lsi_info;
    struct LDAP_Server_Info *prev;
    struct LDAP_Server_Info *next;
} LDAPServerInfo;
```

where:

lsi_host
Returns the fully qualified host name for the LDAP server as a null-terminated string.

lsi_port
Returns the port number assigned to the LDAP server.

lsi_suffix
Returns the naming context for the LDAP server as a null-terminated string. This field is NULL if there is no published naming context for the LDAP server.

lsi_query_key
Returns the service name as a null-terminated string. The service name is formed by concatenating the service key and an optional eNetwork domain name. For example, if the service key is `ldap` and the eNetwork domain name is `research`, the service name is `ldap.research`.

lsi_dns_domain
Returns the DNS domain where the LDAP server information was published. This is a null-terminated string.

lsi_replica_type
Returns the LDAP server type and is set to one of the following values:

LDAP_LSI_MASTER
The server is a master.

LDAP_LSI_REPLICA
The server is a replica.

LDAP_LSI_NO_SERVER_TYPE
The server type is not known.

lsi_sec_type
Returns the connection security type and is set to one of the following values:

LDAP_LSI_NOSSL
The connection is non-SSL.

LDAP_LSI_SSL
The connection is SSL.

LDAP_LSI_NO_SECURITY_TYPE
The security type is not known.

lsi_priority
Returns the priority value for the LDAP server. The `LDAPServerInfo` list entries are ordered based on the priority value such that entries with smaller priority values are listed before entries with larger priority values.

lsi_weight
Returns the weight value for the LDAP server. The `LDAPServerInfo` list entries are load-balanced within a priority class based on the weight value such that entries with larger weight values are more likely to be listed before entries with smaller weight values

lsi_vendor_info
Returns the vendor information for the LDAP server. This is a null-terminated string. This field is `NULL` if there is no published vendor information for the LDAP server.

lsi_info
Returns the general information for the LDAP server. This is a null-terminated string. This field is `NULL` if there is no published general information for the LDAP server.

prev
The address of the previous entry in the server information list. This field is `NULL` if this is the first entry in the list.

next
The address of the next entry in the server information list. This field is `NULL` if this is the last entry in the list.

In general, an application can locate a suitable LDAP server as follows:

1. Before connecting to an LDAP server in the enterprise, the application should call the **ldap_server_locate()** routine to obtain a list of one or more LDAP servers that have been published in DNS or in the server information file. (For details about information contained in the server information file, see "LDAP server information file" on page 246.) The application can normally use the default request settings by specifying `NULL` for the *server_request* parameter. If the application does not specify *search_source*, the **ldap_server_locate()** routine looks for server information in the server information file and then uses DNS if the server information file does not exist, if the server information file entries have expired, or if no servers in the server information file match the search criteria. If no server entries are found and the application does not specify the service key (which defaults to `ldap`), the **ldap_server_locate()** routine retries the search using `_ldap` for the service key.
2. Once the application has obtained the list of servers, it should walk the list, using the first server that meets its needs. This maximizes the advantage that can be derived from using the priority and weighting scheme implemented by the administrator. The application might not want to use the first server in the list for several reasons:
   a. The client must specifically connect using SSL or non-SSL. The *lsi_sec_type* field in the `LDAPServerInfo` entry is set to `LDAP_LSI_SSL` if the server is listening for an SSL connection and to `LDAP_LSI_NOSSL` if the server is listening for a non-SSL connection. This field is set based on the `service` entry supplied by the administrator in the `DNS` TXT record for the LDAP server. If an LDAP server accepts both SSL and non-SSL connections, the administrator should define two TXT records for the server, one specifying "**service:ldap://***host:port*/" and the other specifying "**service:ldaps://***host:port*/". This results in two `LDAPServerInfo` entries for the LDAP server, one specifying `LDAP_LSI_NOSSL` and the other specifying `LDAP_LSI_SSL`.

      The *lsi_sec_type* field is set to `LDAP_LSI_NO_SECURITY_TYPE` if the administrator did not specify a `service` TXT record for the LDAP server. In this case, the application can query the root DSE to determine if the server supports a secure SSL port. This assumes that the LDAP server is listening on a port that is known to the application (for example, the default port of 389).
   b. The client must connect to a master or replica. The *lsi_replica_type* field in the `LDAPServerInfo` entry is set to `LDAP_LSI_MASTER` if the LDAP server is a master and to `LDAP_LSI_REPLICA` if the server is a replica. This field is set based on the `ldaptype` entry supplied by the administrator in the `DNS` TXT record for the LDAP server. The *lsi_replica_type* field is set to `LDAP_LSI_NO_SERVER_TYPE` if the administrator did not specify an `ldaptype` TXT record.
   c. The client must connect to a server that supports a particular naming context. Note the list of servers returned in the list can be filtered by specifying a value for the *DN_filter* field in the `LDAPServerRequest`, which filters out servers that do not have a naming context under which the DN resides. The naming contexts supported by the LDAP server are obtained from the `service` TXT records for the LDAP server. The application can query the root DSE to determine the supported naming contexts if the administrator did not provide `service` TXT records containing the naming contexts.
3. Once the client has selected a server, it calls the **ldap_init()** or **ldap_ssl_init()** routine. If the selected server is unavailable, the application should continue

processing the server list returned by the **ldap_server_locate()** routine until an available server is found or the list is exhausted.

The resource name for LDAP servers published in DNS is formed by combining the service key, eNetwork domain name, protocol key, and domain name as follows:

service-key.eNetwork-domain.protocol-key.domain-name

Example: If the service key is `_ldap`, the eNetwork domain is `marketing`, the protocol key is `_tcp`, and the domain name is `mycorp.com`, the resource name for the DNS SRV record would be:

```
_ldap.marketing._tcp.mycorp.com
```

If no eNetwork domain is specified, the resource name would be:

```
_ldap._tcp.mycorp.com
```

Function return value

The function return value is `LDAP_SUCCESS` if no error is detected. Otherwise, it is one of the LDAP error codes listed in the `ldap.h` include file.

The following are some common errors for this routine:

LDAP_DNS_CONF_FILE_ERROR
Server information file error.

LDAP_DNS_CONF_FILE_EXPIRED
Server information file is expired and `LDAP_LSI_CONF_ONLY` is specified for the search source.

LDAP_DNS_INVALID_DATA
Name server response is not valid.

LDAP_DNS_NO_SERVERS
No LDAP servers are available.

LDAP_DNS_TRUNCATED
Name server response is truncated and TCP connections are not available.

LDAP_LOCAL_ERROR
A system routine detected an error.

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_PARAM_ERROR
An incorrect request parameter is specified. This error can occur if the generated DNS resource name is longer than 255 characters.

ldap_set_option(), ldap_set_option_np()

Purpose

Set the value for an LDAP option

Format

```
#include <ldap.h>

int ldap_set_option(
    LDAP *                ld,
    int                   option,
    void *                value)

int ldap_set_option_np(
    LDAP *                ld,
    int                   option,
    ...)
```

Parameters

Input

ld Specifies the LDAP handle.

option
Specifies the option identifier.

value
Specifies the option value.

Usage

The **ldap_set_option()** and **ldap_set_option_np()** routines set the value for an LDAP option in the supplied LDAP handle. The routines differ only in the way the third parameter is specified.

The manner in which the LDAP option value is specified for the **ldap_set_option()** routine depends upon the LDAP protocol version option for the LDAP handle. The manner in which the LDAP option value is specified for the **ldap_set_option_np()** routine is not dependent upon the LDAP protocol version option for the LDAP handle. Note the default LDAP protocol version is 2 for LDAP handles created by the **ldap_open()** routine and 3 for LDAP handles created by the **ldap_init()** and **ldap_ssl_init()** routines. Table 4 summarizes how to specify the options.

Table 4. How to specify options for the ldap_set_option and ldap_set_option_np routines

Option	ldap_set_option Version 2	ldap_set_option Version 3	ldap_set_option_np
LDAP_OPT_CLIENT_CONTROLS		LDAPControl **	LDAPControl **
LDAP_OPT_DEBUG	int	int *	int
LDAP_OPT_DEBUG_FILENAME	char *	char *	char *
LDAP_OPT_DEBUG_STRING	char *	char *	char *
LDAP_OPT_DELEGATION		int	int
LDAP_OPT_DEREF	int	int *	int
LDAP_OPT_EXT_REBIND_FN	LDAPExtRebindProc	LDAPExtRebindProc	LDAPExtRebindProc
LDAP_OPT_IO_CALLBACK	LDAPIOCallback *	LDAPIOCallback *	LDAPIOCallback *
LDAP_OPT_MAX_SASL_LEVEL		int *	int

Table 4. How to specify options for the ldap_set_option and ldap_set_option_np routines (continued)

Option	ldap_set_option Version 2	ldap_set_option Version 3	ldap_set_option_np
LDAP_OPT_MIN_SASL_LEVEL		int *	int
LDAP_OPT_PROTOCOL_VERSION	int *	int *	int
LDAP_OPT_REBIND_FN	LDAPRebindProc	LDAPRebindProc	LDAPRebindProc
LDAP_OPT_REFERRALS	int	int	int
LDAP_OPT_REFHOPLIMIT	int	int *	int
LDAP_OPT_RESTART	int	int	int
LDAP_OPT_SERVER_CONTROLS		LDAPControl **	LDAPControl **
LDAP_OPT_SIZELIMIT	int	int *	int
LDAP_OPT_SOCKS_CONF	char *	char *	char *
LDAP_OPT_SOCKS_PASSWORD	char *	char *	char *
LDAP_OPT_SOCKS_SERVER	char *	char *	char *
LDAP_OPT_SOCKS_USERNAME	char *	char *	char *
LDAP_OPT_SOCKS_VERSION	int	int *	int
LDAP_OPT_SSL_CIPHER	char *	char *	char *
LDAP_OPT_SSL_CIPHER_EXPANDED	char *	char *	char *
LDAP_OPT_SSL_CIPHER_FORMAT	int	int	int
LDAP_OPT_SSL_TIMEOUT	int	int *	int
LDAP_OPT_TIMELIMIT	int	int *	int
LDAP_OPT_UTF8_IO	int	int	int
LDAP_OPT_V2_WIRE_FORMAT	int	int	int

Example: The LDAP_OPT_SIZELIMIT option is specified as follows:

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

The following LDAP options can be set:

LDAP_OPT_CLIENT_CONTROLS

The LDAP_OPT_CLIENT_CONTROLS option specifies a default list of client controls to be processed with each request. The end of the list is indicated by a NULL control address. Specify NULL for the list address to clear the current client controls list for the LDAP handle. The entire list is rejected if the list includes a critical client control that is not recognized by the LDAP client run time. A parameter error is returned if the LDAP protocol version is not set to LDAP_VERSION3. The default list can be overridden by specifying a client control, or a list of client controls, on specific API routines. There are no default client controls if the LDAP_OPT_CLIENT_CONTROLS option is not set.

The OID string in the client control is a null-terminated character string in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option of the LDAP handle. In addition, a client control value that is a character string is in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option of the LDAP handle.

The following client controls are supported:

```
ibm-saslBindCramRealmName                1.3.18.0.2.10.12
ibm-saslBindCramUserName                 1.3.18.0.2.10.13
ibm-saslBindDigestRealmName              1.3.18.0.2.10.12
ibm-saslBindDigestUserName               1.3.18.0.2.10.13
ibm-serverHandledSearchRequest           1.3.18.0.2.10.7
```

For more information about client controls, see "Supported client controls" on page 16.

LDAP_OPT_DEBUG

The LDAP_OPT_DEBUG option specifies a bitmap that indicates the level of debug trace you want for the LDAP client run time and overrides the debug trace level that is set by the LDAP_DEBUG environment variable. The debug trace level applies to the entire process and not just the LDAP handle. For this reason, the LDAP handle can be specified as NULL, in which case the **ldap_set_option()** routine expects the debug trace level to be specified as the address of an integer and the **ldap_set_option_np()** routine expects the debug trace level to be specified as an integer. If specified, the LDAP handle must be a valid handle.

The option value is formed by **OR**ing together one or more of the following debug options:

LDAP_DEBUG_ACL
Trace ACL processing

LDAP_DEBUG_ALL
Enable all debug traces (same as LDAP_DEBUG_ANY)

LDAP_DEBUG_ANY
Enable all debug traces (same as LDAP_DEBUG_ALL)

LDAP_DEBUG_ARGS
Trace request arguments

LDAP_DEBUG_BE_CAPABILITIES
Trace backend capabilities

LDAP_DEBUG_BER
Trace ASN.1 encode and decode processing

LDAP_DEBUG_CACHE
Trace cache activity

LDAP_DEBUG_CONNS
Trace connection activity

LDAP_DEBUG_ERROR
Trace errors

LDAP_DEBUG_FILTER
Trace filter processing

LDAP_DEBUG_INFO
Trace informational messages

LDAP_DEBUG_LDAPBE
Trace server backend activity

LDAP_DEBUG_LDBM
Trace file backend activity

LDAP_DEBUG_MESSAGE
Trace message processing

LDAP_DEBUG_MULTISERVER
Trace multiple server activity

LDAP_DEBUG_OFF
Disable all debug traces

LDAP_DEBUG_PACKETS
Trace packet activity

LDAP_DEBUG_PARSE
Trace parsing activity

LDAP_DEBUG_PERFORMANCE
Trace performance statistics

LDAP_DEBUG_PLUGIN
Trace plug-in extension activity

LDAP_DEBUG_REFERRAL
Trace referral activity

LDAP_DEBUG_REPLICATION
Trace replication activity

LDAP_DEBUG_SCHEMA
Trace schema processing

LDAP_DEBUG_SDBM
Trace RACF backend activity

LDAP_DEBUG_STATS
Trace operational statistics

LDAP_DEBUG_STRBUF
Trace and UTF-8 activity

LDAP_DEBUG_SYSPLEX
Trace sysplex activity

LDAP_DEBUG_TDBM
Trace TDBM database processing

LDAP_DEBUG_THREAD
Trace thread activity

LDAP_DEBUG_TRACE
Trace API routine entry and exit

Note some of these trace points are applicable only for the LDAP server and do not generate any trace output for the LDAP client. For more information about the LDAP trace options, see “Enabling tracing” on page 242.

LDAP_OPT_DEBUG_FILENAME
The `LDAP_OPT_DEBUG_FILENAME` option specifies the name of the LDAP trace output file and overrides the name that is set by the `LDAP_DEBUG_FILENAME` environment variable. The debug file name applies to the entire process and not just the LDAP handle. For this reason, the LDAP handle can be specified as `NULL`. If specified, the LDAP handle must be a valid handle. The file name is in the local EBCDIC code page or UTF-8, as determined by the `LDAP_LIBASCII` compiler variable.

The trace output is written to stdout if the LDAP_OPT_DEBUG_FILENAME option is not set and the LDAP_DEBUG_FILENAME environment variable is not defined. Therefore, the LDAP_OPT_DEBUG_FILENAME option should be set before either the LDAP_OPT_DEBUG or LDAP_OPT_DEBUG_STRING option is set if the trace output is not to be written to the default trace file as specified by the LDAP_DEBUG_FILENAME environment variable.

The current process identifier is included as part of the trace file name when the name contains a percent sign (%). For example, if LDAP_OPT_DEBUG_FILENAME is set to /tmp/ldap.%.trc and the current process identifier is 247, then the trace file name is /tmp/ldap.247.trc. The trace file name should be unique for each process with LDAP trace enabled because the trace output can be corrupted if multiple processes use the same trace file.

LDAP_LOCAL_ERROR is returned if the specified trace file cannot be opened. In this case, the trace output is written to stdout until a subsequent call is successful in setting the LDAP_OPT_DEBUG_FILENAME option.

LDAP_OPT_DEBUG_STRING

The LDAP_OPT_DEBUG_STRING option specifies LDAP trace options as a null-terminated character string and either completely replaces or incrementally modifies the trace options that are set by the LDAP_DEBUG environment variable. The debug trace level applies to the entire process and not just the LDAP handle. For this reason, the LDAP handle can be specified as NULL. If specified, the LDAP handle must be a valid handle. The debug string is in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable.

The value for LDAP_OPT_DEBUG_STRING is a character string that can be specified as follows:

- A decimal value (for example, 32)
- A hexadecimal value (for example, x20 or X20)
- A keyword (for example, FILTER)
- A construct of these values using plus and minus signs to indicate inclusion or exclusion of a value.

The trace options that are specified by the LDAP_DEBUG environment variable are modified if the LDAP_OPT_DEBUG_STRING starts with a plus or minus sign. Otherwise, the trace options that are specified by the LDAP_DEBUG environment variable are replaced with the options specified by the LDAP_OPT_DEBUG_STRING option. For more information about the LDAP trace options, see “Enabling tracing” on page 242.

LDAP_OPT_DELEGATION

The LDAP_OPT_DELEGATION option specifies whether the LDAP client passes Kerberos delegated credentials to the LDAP server. It must be set to either LDAP_OPT_ON or LDAP_OPT_OFF. The default is LDAP_OPT_OFF. A parameter error is returned if the LDAP protocol version is not set to LDAP_VERSION3. Use this option if you want to allow the LDAP server to use the client's credentials for requests. Note the server might or might not support this capability.

LDAP_OPT_DEREF

The LDAP_OPT_DEREF option specifies how the LDAP server handles aliases during search request. It must have one of the following values:

LDAP_DEREF_ALWAYS
Dereference aliases both in searching and in locating the base object of the search.

LDAP_DEREF_FINDING
Dereference aliases in locating the base object of the search but not when searching subordinates of the base object.

LDAP_DEREF_NEVER
Do not dereference aliases. (This is the default.)

LDAP_DEREF_SEARCHING
Dereference aliases in subordinates of the base object in searching but not in locating the base object of the search.

LDAP_OPT_EXT_REBIND_FN
The LDAP_OPT_EXT_REBIND_FN option specifies the routine to be called by the LDAP client run time when it must authenticate a connection with another LDAP server. This can occur when the LDAP client is following a referral returned by the initial LDAP server. If a rebind routine is not defined, referrals are followed using an anonymous bind. For more information about the rebind routine, see "Rebinding while following referrals" on page 12. Specify NULL for the rebind function to stop using a rebind routine.

The rebind routine set by the LDAP_OPT_EXT_REBIND_FN option can be used if both LDAP_OPT_EXT_REBIND_FN and LDAP_OPT_REBIND_FN are set for the LDAP handle.

LDAP_OPT_IO_CALLBACK
The LDAP_OPT_IO_CALLBACK option specifies routines to be called by the LDAP client run time when it must communicate with the LDAP server. The C/C++ runtime (LE) socket routines, such as **socket()**, **bind()**, **connect()**, **getpeername()**, **send()**, **select()**, **recv()**, and **close()**, are used if the application does not provide its own routines.

The LDAP_OPT_IO_CALLBACK option cannot be changed after a connection is established with the LDAP server. Specify NULL for the address of the LDAPIOCallback structure to revert to the normal socket routines. The callback routines are used when following referrals returned by the local LDAP server. The LDAPIOCallback structure is defined as follows:

```
typedef struct _LDAPIOCallback {
  void *  userData;
  int     (*connect)(const char * host, int port,
                     int * desc, void * userData);
  int     (*getpeer)(int desc, struct sockaddr * addr,
                     size_t size, size_t * length,
                     void * userData);
  int     (*send)(int desc, const void * buffer, size_t length,
                  void * userData);
  int     (*select)(int desc[], struct timeval * timeout,
                    int * rtndesc, void * userData);
  int     (*recv)(int desc, void * buffer, size_t size,
                  size_t * length, void * userData);
  void    (*close)(int desc, void * userData);
} LDAPIOCallback;
```

The fields in the LDAPIOCallback structure are used as follows:

userData
The *userData* value is passed to each of the callback routines. Specify NULL for this field if you do not need to pass anything to the callback routines.

connect
The *connect* routine should be called when the LDAP client run time must establish a connection with the LDAP server. The *host* and *port* values are obtained from the **ldap_init()**, **ldap_ssl_init()** or **ldap_open()** routine. If an LDAP URL was specified, the *host* parameter contains the host name that is obtained from the URL. The callback routine can use these values to establish the connection, or can ignore them and use a different algorithm to determine the target for the connection. The return value must be `0` if the connection is successful, or a value that is defined in `errno.h` if the connection is unsuccessful. The *desc* parameter should be set to a descriptor for the connection if the request is successful. The descriptor can be anything that is meaningful to the application if it is not `-1`. The descriptor is passed to the other callback routines.

getpeer The *getpeer* routine should be called to obtain the connection name for the LDAP server. For a TCP/IP-based connection, this should be a `struct sockaddr` for the AF_INET or AF_INET6 family. The *addr* and *size* parameters identify the address and size of the return buffer. The callback routine should set the *length* parameter to the actual size of the returned identification. The return value should be `0` for a normal return, or a value that is defined in `errno.h` for a failure return.

send The *send* routine should be called to send data to the LDAP server. The callback routine is responsible for ensuring that all the data is sent to the LDAP server (that is, this is a blocking send). The *buffer* and *length* parameters identify the data to be sent. The return value should be `0` if the data is sent, or a value that is defined in `errno.h` if the data cannot be sent.

select The *select* routine should be called to wait for data on one or more LDAP server connections. The *desc* parameter is an array of descriptors with the last entry in the array set to `-1`. The *timeout* parameter specifies how long to wait for data to become available. `NULL` is passed for the *timeout* parameter if the select routine is to wait indefinitely. The return value should be `EAGAIN` if the time limit is reached, `EINTR` if the wait is interrupted by a signal, or `0` if there is data or status available for a connection. The *rtndesc* parameter should be set to the descriptor with pending data or status.

recv The *recv* routine should be called to receive data or connection status from the LDAP server. The callback routine should not return until it has either data or an error (that is, this is a blocking receive). The *buffer* and *size* parameters identify the receive buffer address and size. The callback routine should set the *length* parameter to the actual data length. The return value should be `0` if data is received, `ECONNRESET` if the connection is closed, or a value that is defined in `errno.h` if an error is detected.

close The *close* routine should be called to close the connection to the LDAP server.

LDAP_OPT_MAX_SASL_LEVEL
The `LDAP_OPT_MAX_SASL_LEVEL` option specifies the maximum SASL protection level for the LDAP handle. This is the highest SASL protection level that can be negotiated during a bind using a `SASL` mechanism. The negotiated protection level cannot be greater than this level even if the

server offers a higher protection level. LDAP_PARAM_ERROR is returned if the LDAP protocol version is not set to LDAP_VERSION3.

The SASL protection levels, in increasing level of protection, are:

LDAP_SASL_LEVEL_NONE
No integrity or confidentiality protection.

LDAP_SASL_LEVEL_INTEG
Integrity protection.

LDAP_SASL_LEVEL_CONF
Integrity and confidentiality protection. (This is the default.)

LDAP_OPT_MIN_SASL_LEVEL
The LDAP_OPT_MIN_SASL_LEVEL option specifies the minimum SASL protection level for the LDAP handle. This is the lowest SASL protection level that can be negotiated during a bind using a SASL mechanism. The bind fails if the server does not offer at least this protection level. LDAP_PARAM_ERROR is returned if the LDAP protocol version is not set to LDAP_VERSION3.

The SASL protection levels, in increasing level of protection, are:

LDAP_SASL_LEVEL_NONE
No integrity or confidentiality protection. (This is the default.)

LDAP_SASL_LEVEL_INTEG
Integrity protection.

LDAP_SASL_LEVEL_CONF
Integrity and confidentiality protection.

LDAP_OPT_PROTOCOL_VERSION
The LDAP_OPT_PROTOCOL_VERSION option specifies the LDAP protocol version that is used by the LDAP client when connecting to an LDAP server. It must be set to either LDAP_VERSION2 or LDAP_VERSION3. The default is LDAP_VERSION3 if **ldap_init()** or **ldap_ssl_init()** can be used to create the LDAP handle, and LDAP_VERSION2 if **ldap_open()** can be used to create the LDAP handle. In either case, the LDAP_OPT_PROTOCOL_VERSION option can be used to change the default protocol version. The protocol version must be set before the client binds to an LDAP server as a result of calling **ldap_bind()**, **ldap_bind_s()**, **ldap_sasl_bind()**, **ldap_sasl_bind_s()**, **ldap_simple_bind()**, **ldap_simple_bind_s()**, or any routine that causes an implicit bind. An error is returned if the LDAP_OPT_PROTOCOL_VERSION option is specified after a connection is established with the LDAP server.

Note: The LDAP protocol version affects the way parameters are specified for the **ldap_set_option()** routine. Therefore, the LDAP_OPT_PROTOCOL_VERSION option should be set before any other LDAP options are set.

LDAP_OPT_REBIND_FN
The LDAP_OPT_REBIND_FN option specifies the routine to be called by the LDAP client run time when it must authenticate a connection with another LDAP server. This can occur when the LDAP client is following a referral that is returned by the initial LDAP server. If a rebind routine is not defined, referrals are followed using an anonymous bind. For more information about the rebind routine, see “Rebinding while following referrals” on page 12. Specify NULL for the rebind function to stop using a rebind routine.

The rebind routine set by the LDAP_OPT_EXT_REBIND_FN option can be used if both LDAP_OPT_EXT_REBIND_FN and LDAP_OPT_REBIND_FN are set for the LDAP handle.

LDAP_OPT_REFERRALS
The LDAP_OPT_REFERRALS option specifies whether the LDAP client follows referrals that are returned by the LDAP server. It must be set to either LDAP_OPT_ON or LDAP_OPT_OFF. The default is LDAP_OPT_ON.

LDAP_OPT_REFHOPLIMIT
The LDAP_OPT_REFHOPLIMIT option specifies the maximum number of LDAP servers to contact when following a referral. For subtree searches, this is the limit on the depth of nested search references, so the number of servers that are contacted might actually exceed this value. The default is 10.

LDAP_OPT_RESTART
The LDAP_OPT_RESTART option specifies whether the **select()** system call should be restarted when it is interrupted by the system. It must be set to either LDAP_OPT_ON or LDAP_OPT_OFF. The default is LDAP_OPT_OFF.

LDAP_OPT_SERVER_CONTROLS
The LDAP_OPT_SERVER_CONTROLS option specifies a default list of server controls to be sent with each request. The end of the list is indicated by a NULL control address. Specify NULL for the list address to clear the current server controls list for the LDAP handle. A parameter error is returned if the LDAP protocol version is not set to LDAP_VERSION3. The default list can be overridden by specifying a server control, or a list of server controls, on specific API routines. There are no default server controls if the LDAP_OPT_SERVER_CONTROLS option is not set.

The OID string in the server control is a null-terminated character string in UTF-8 or the local EBCDIC code page, as determined by the LDAP handle. The OID value is assumed to already be in the correct format for transmission to the server and the LDAP client does not modify it.

LDAP_OPT_SIZELIMIT
The LDAP_OPT_SIZELIMIT option specifies the maximum number of entries that can be returned for a search request. The LDAP server can also provide a size limit on the number of entries returned. For information about the server's size limit and how it interacts with the client size limit, see the documentation for your LDAP server. For the IBM Tivoli Directory Server for z/OS, see the description of the **sizeLimit** configuration file option (Customizing the LDAP server configuration) in *z/OS IBM Tivoli Directory Server Administration and Use for z/OS*. The default size limit for the client, which is specified by a value of 0, indicates that the maximum number of entries is limited only by the LDAP server limit.

LDAP_OPT_SOCKS_CONF
The LDAP_OPT_SOCKS_CONF option specifies the name of the SOCKS configuration file to be used when connecting to the LDAP server, and overrides the SOCKS_CONF and SOCKS_SERVER environment variables and also the LDAP_OPT_SOCKS_SERVER option. The option value is a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable. Specify NULL for the option value to cancel the SOCKS configuration that is specified by the SOCKS_CONF or SOCKS_SERVER environment variable and use a direct connection to the LDAP server.

LDAP_OPT_SOCKS_PASSWORD
The LDAP_OPT_SOCKS_PASSWORD option specifies the SOCKS password to be

used when connecting to the LDAP server through a SOCKS server, and overrides the SOCKS_PASSWORD environment variable. The option value is a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable. Specify NULL for the option value to indicate that no password is to be used.

A SOCKS user name and password are required when using the SOCKS version 5 protocol and the SOCKS server is configured to require user authentication. An unauthenticated SOCKS connection can be used if the SOCKS user name and password are not set. Note authentication for the SOCKS connection is separate from the bind authentication for the LDAP server. The SOCKS user name and password are not used for the SOCKS version 4 protocol.

LDAP_OPT_SOCKS_SERVER

The LDAP_OPT_SOCKS_SERVER option specifies the SOCKS servers to be used when connecting to the LDAP server, and overrides the SOCKS_CONF and SOCKS_SERVER environment variables and also the LDAP_OPT_SOCKS_CONF option. The option value is a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable, and consists of a comma-separated list of SOCKS servers. Each SOCKS server is specified as *host:port*. The *host* is a DNS name, an IPv4 address in dotted decimal format, or an IPv6 address in colon-separated format that is enclosed in square brackets. The *port* defaults to 1080, if it is not specified. Specify NULL for the option value to cancel the SOCKS configuration that is specified by the SOCKS_CONF or SOCKS_SERVER environment variable and use a direct connection to the LDAP server.

LDAP_OPT_SOCKS_USERNAME

The LDAP_OPT_SOCKS_USERNAME option specifies the SOCKS user name to be used when connecting to the LDAP server through a SOCKS server. It overrides the SOCKS_USERNAME environment variable. The option value is a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable. Specify NULL for the option value to indicate that no user name is to be used.

A SOCKS user name and password are required when using the SOCKS version 5 protocol and the SOCKS server is configured to require user authentication. An unauthenticated SOCKS connection can be used if the SOCKS user name and password are not set. Note authentication for the SOCKS connection is separate from the bind authentication for the LDAP server. The SOCKS user name and password are not used for the SOCKS version 4 protocol.

LDAP_OPT_SOCKS_VERSION

The LDAP_OPT_SOCKS_VERSION option specifies the SOCKS protocol version, and overrides the SOCKS_VERSION environment variable. The valid values are 4 and 5. The default is 4. However, the SOCKS version 5 protocol is always used when the LDAP server address is an IPv6 address because the SOCKS version 4 protocol does not support IPv6 addresses. You can set the LDAP_OPT_SOCKS_VERSION option to 5 to cause the LDAP client run time to always use the SOCKS version 5 protocol.

LDAP_OPT_SSL_CIPHER

This option is pertinent if 2-byte SSL ciphers are currently in effect, which is based on the setting of LDAP_OPT_SSL_CIPHER_FORMAT or the

LDAP_OPT_SSL_CIPHER_FORMAT option. Also see the description of the option LDAP_OPT_SSL_CIPHER_EXPANDED below. Only one of these settings are pertinent at one time.

The LDAP_OPT_SSL_CIPHER option specifies one or more cipher suites to be used when negotiating an SSL connection with the LDAP server. The default SSL cipher suites are used if the LDAP_OPT_SSL_CIPHER option is not set. The GSK_V3_CIPHER_SPECS environment variable can be used to change the default cipher suites. The option value is a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable. The string consists of the cipher suites you want as two hexadecimal digits per cipher suite. For example, to choose from RC4-MD5-US, RC4-SHA-1, and AES-128-SHA-1, specify 04052F.

For compatibility with prior releases, the ldap.h include file provides definitions for 2-byte cipher suites that are supported in SSL V3 and TLS V1.0 as a coding convenience. The definitions are summarized in Table 3. The mnemonics ending in _EX are always available. The other mnemonics are available only when the SSL Security Level 3 FMID is installed. For newer cipher suites supported in later TLS protocols, use the two hexadecimal digit forms for each cipher suite you want. For more information about the GSK V3_CIPHER_SPECS environment variable and SSL cipher suites, see and in *z/OS Cryptographic Services System SSL Programming*. The SSL cipher list must be set before an SSL connection is established to the LDAP server.

LDAP_OPT_SSL_CIPHER_EXPANDED

This option is pertinent provided 4-byte SSL ciphers are currently in effect, which is based on the setting of the environment variable LDAP_SSL_CIPHER_FORMAT or the LDAP_OPT_SSL_CIPHER_FORMAT option. Also, see the description of the option LDAP_OPT_SSL_CIPHER above. Only one of these two settings are pertinent at one time.

The LDAP_OPT_SSL_CIPHER option specifies one or more cipher suites to be used when negotiating an SSL connection with the LDAP server. The default SSL cipher suites are used if the LDAP_OPT_SSL_CIPHER_EXPANDED option is not set. The GSK_V3_CIPHER_SPECS_EXPANDED environment variable can be used to change the default cipher suites. The option value is a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable. The string consists of the cipher suites you want as four hexadecimal digits per cipher suite. For example, to choose from RC4-MD5-US, RC4-SHA-1, and AES-128-SHA-1, specify 00040005002F.

For more information about the GSK_V3_CIPHER_SPECS_EXPANDED environment variable and SSL cipher suites, see and in *z/OS Cryptographic Services System SSL Programming*. The SSL cipher list must be set before an SSL connection is established to the LDAP server.

LDAP_OPT_SSL_CIPHER_FORMAT

This option indicates the length of SSL cipher specifications to be used. Valid values are LDAP_SSL_CIPHER_FORMAT_CHAR2 and LDAP_SSL_CIPHER_FORMAT_CHAR4, where LDAP_SSL_CIPHER_FORMAT_CHAR2 is the default setting.

A value of LDAP_SSL_CIPHER_FORMAT_CHAR2 indicates the cipher suites in use come from either SSL defaults, as determined from the

GSK_V3_CIPHER_SPECS environment variable, or from a setting of LDAP_OPT_SSL_CIPHER, by way of the **ldap_set_option()** routine.

A value of LDAP_SSL_CIPHER_FORMAT_CHAR4 indicates the cipher suites in use come from either SSL defaults, as determined from the GSK_V3_CIPHER_SPECS_EXPANDED environment variable, or from a setting of LDAP_OPT_SSL_CIPHER_EXPANDED, by way of the **ldap_set_option()** routine.

The SSL cipher list must be set before an SSL connection is established to the LDAP server.

LDAP_OPT_SSL_TIMEOUT

The LDAP_OPT_SSL_TIMEOUT option specifies the SSL session timeout value in seconds. Cached SSL sessions are discarded after the specified number of seconds. Cached SSL sessions can be reused and improve performance by eliminating the need for a full SSL handshake when reconnecting to an LDAP server. SSL sessions are not cached if the timeout value is zero. If the LDAP_OPT_SSL_TIMEOUT option is not set, the default SSL session timeout of 86400 seconds can be used. The GSK_V3_SESSION_TIMEOUT environment variable can be used to change the default SSL session timeout value. The SSL timeout value must be set before an SSL connection is established to the LDAP server. The LDAP_OPT_SSL_TIMEOUT option is ignored if the **ldap_ssl_client_init()** routine should be called to initialize the SSL environment.

LDAP_OPT_TIMELIMIT

The LDAP_OPT_TIMELIMIT option specifies the number of seconds to wait for search results. The LDAP server can also provide a limit on the search time. For information about the server's search time limit and how it interacts with the client time limit, see the documentation for your LDAP server. For the IBM Tivoli Directory Server for z/OS, see the description of the **timeLimit** configuration file option (Customizing the LDAP server configuration) in *z/OS IBM Tivoli Directory Server Administration and Use for z/OS*. The default time limit for the client, which is specified by a value of 0, indicates that there is no client time limit and that the maximum number of seconds is limited only by the LDAP server limit.

LDAP_OPT_UTF8_IO

The LDAP_OPT_UTF8_IO option specifies the format of text data that is provided as input to an LDAP API routine or returned as output by an LDAP API routine. LDAP_OPT_ON indicates text data is in the UTF-8 code set. LDAP_OPT_OFF indicates text data is in the code set of the current locale. The default is LDAP_OPT_ON if the LDAP_LIBASCII compiler variable is defined and LDAP_OPT_OFF otherwise.

The LDAP_OPT_UTF8_IO option applies to all LDAP API routines that accept an LDAP handle as an input parameter, unless noted otherwise in the description of the API routine. Text data for LDAP API routines that do not accept an LDAP handle as an input parameter is in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable.

LDAP_OPT_V2_WIRE_FORMAT

The LDAP_OPT_V2_WIRE_FORMAT option specifies the format of attribute values that are exchanged between the LDAP client and the LDAP server using the LDAP version 2 protocol. (Attribute values that are exchanged by using the LDAP version 3 protocol are always in UTF-8.)

LDAP_OPT_V2_WIRE_FORMAT_ISO8859_1 indicates attribute values are exchanged using the ISO8859-1 code page. LDAP_OPT_V2_WIRE_FORMAT_UTF8 indicates attribute values are exchanged using UTF-8. The default is LDAP_OPT_V2_WIRE_FORMAT_UTF8.

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes that are listed in the ldap.h include file.

The following are some common errors for this routine:

LDAP_INVALID_STATE
The LDAP handle is not in the correct state for the requested operation.

LDAP_LOCAL_ERROR
A system routine returned an error.

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_PARAM_ERROR
A parameter is not valid or the LDAP protocol version is not correct for the requested option.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION
A critical client control is not recognized.

ldap_set_rebind_proc()

Purpose

Specify the routine to be called when binding to another LDAP server

Format

```
#include <ldap.h>

int ldap_set_rebind_proc(
    LDAP *                  ld,
    LDAPRebindProc          proc)
```

Parameters

Input

ld Specifies the LDAP handle.

proc
Specifies the routine to be called.

Usage

The **ldap_set_rebind_proc()** routine specifies the routine to be called by the LDAP client run time when it must authenticate a connection with another LDAP server. This occurs when the LDAP client is following a referral returned by an LDAP server. If a rebind routine is not defined, referrals are followed using an anonymous bind.

For more information about the rebind routine, see "Rebinding while following referrals" on page 12. You can set the rebind routine either by calling **ldap_set_rebind_proc()** or by calling **ldap_set_option()** to set the LDAP_OPT_REBIND_FN option. The rebind routine that can be used is the last one set by either method. The **ldap_set_rebind_proc()** routine cannot be used to specify an extended bind routine. To specify an extended bind, use the **ldap_set_option()** routine to set the LDAP_OPT_EXT_REBIND_FN option.

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes listed in the ldap.h include file.

The following is a common error for this routine:

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_simple_bind(), ldap_simple_bind_s()

Purpose

Bind to the LDAP server using a distinguished name (DN) and password

Format

```
#include <ldap.h>

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

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

Parameters

Input

ld Specifies the LDAP handle.

who
Specifies the distinguished name as a null-terminated character string. The distinguished name is in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. An anonymous bind is performed if this parameter is NULL or the distinguished name is a zero-length string.

passwd
Specifies the password as a null-terminated character string. The password is in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle.

Usage

The **ldap_simple_bind()** or **ldap_simple_bind_s()** routine binds to the LDAP server identified by the LDAP handle. The LDAP server authenticates the client using the distinguished name and password. Note this information is sent unencrypted to the LDAP server unless an SSL connection can be used.

The **ldap_simple_bind()** routine sends the bind message to the LDAP server and returns control to the application. The application should call the **ldap_result()** routine to get the response to the bind request.

The **ldap_simple_bind_s()** routine sends the bind message to the LDAP server and waits for a response. The bind request is abandoned if the client is unable to wait for the response because of an error from the **ldap_result()** routine.

Client controls specified by the LDAP_OPT_CLIENT_CONTROLS and server controls specified by the LDAP_OPT_SERVER_CONTROLS options are used by the **ldap_simple_bind()** and **ldap_simple_bind_s()** routines.

Function return value

The function return value for the **ldap_simple_bind()** routine is the message identifier of the bind message, or -1 if a client error occurred. When the return value is -1, the application should call the **ldap_get_errno()** routine to get the LDAP error code. Any errors reported by the LDAP server are not returned by the **ldap_simple_bind()** routine. Instead, the application must call the **ldap_parse_result()** routine to obtain the result code from the bind response message returned by the **ldap_result()** routine.

The function return value for the **ldap_simple_bind_s()** routine is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes listed in the ldap.h include file. Errors reported by the LDAP server are returned by the **ldap_simple_bind_s()** routine including errors detected by the LDAP client.

The following are some common client errors:

LDAP_INVALID_STATE
A bind or unbind is in progress for the LDAP handle or an application exit is active for the LDAP handle.

LDAP_LOCAL_ERROR
A system function reported an error.

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_PARAM_ERROR
A parameter is not valid.

LDAP_SERVER_DOWN
Unable to connect to LDAP server.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION
A critical client control is either not recognized or is not supported for a bind operation.

The following are some common bind result codes:

LDAP_INAPPROPRIATE_AUTH
Inappropriate authentication provided by the client.

LDAP_INVALID_CREDENTIALS
The credentials provided by the client are not valid.

LDAP_REFERRAL
The server cannot accept the bind.

LDAP_STRONG_AUTH_REQUIRED
Strong authentication is required by the server.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION
A critical server control is either not recognized or is not supported for a bind operation.

ldap_ssl_client_init()

Purpose

Initialize the SSL client run time

Format

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

int ldap_ssl_client_init(
    const char *          keyring,
    const char *          keyring_pw,
    int                   ssl_timeout,
    int *                 ssl_rsncode)
```

Parameters

Input

keyring
Specifies the name of the SSL key database, SAF key ring, or PKCS #11 token as a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable. Specify NULL for this parameter to use the GSK_KEYRING_FILE environment variable. An SSL key database must be a z/OS UNIX System Services file and cannot be a partitioned or sequential data set. For a PKCS #11 token, specify the following format to indicate the token to be used:

TOKEN/*NAME*

where *NAME* is the name of the PKCS #11 token.

keyring_pw
Specifies the password for the SSL key database as a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable. Specify **file://***filename* to use an SSL stash file where *filename* is the name of the stash file. Specify a zero-length character string to use a SAF key ring or PKCS #11 token instead of a key database. Specify NULL for this parameter to use the GSK_KEYRING_PW or GSK_KEYRING_STASH environment variable. An SSL stash file must be a z/OS UNIX System Services file and cannot be a partitioned or sequential data set. If NULL is specified and the GSK_KEYRING_PW and GSK_KEYRING_STASH environment variables are not defined, a SAF key ring or PKCS #11 token can be used. The PKCS #11 token is used if the *keyring* parameter is in the following format:

TOKEN/*NAME*

If NULL is specified for the *keyring* parameter, this parameter is ignored.

ssl_timeout
Specifies the SSL session cache timeout in seconds. The value must be between 1 and 86400. Specify a value of 0 to use the GSK_V3_SESSION_TIMEOUT environment variable. If 0 is specified and the GSK_V3_SESSION_TIMEOUT environment variable is not defined, the default is 86400.

Output

ssl_rsncode
Returns the LDAP reason code as defined in the ldapssl.h include file. Specify NULL for this parameter if the LDAP reason code is not needed.

Usage

The **ldap_ssl_client_init()** routine initializes the SSL client run time and must be called before any SSL options are set or an SSL connection is established with an LDAP server. In addition, **ldap_ssl_client_init()** must be run before starting **ldap_init()** or **ldap_ssl_init()** to create a handle for an SSL connection. An error is returned if **ldap_ssl_client_init()** should be called more than once. LDAP does not support SSL V2 protocol, and disables it from being used. SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 protocols are supported. The z/OS System SSL defaults and environment variables control which of these supported protocols are enabled or disabled. For example, the environment variable GSK_PROTOCOL_SSLV3 can be set to "ON" to enable SSL V3 protocol, or "OFF" to disable SSL V3 protocol. The environment variable GSK_PROTOCOL_TLSV1 can be set to "ON" to enable TLS V1.0 protocol, or "OFF" to disable TLS V1.0 protocol. TLS V1.1 and TLS V1.2 protocols are disabled by default. To enable TLS V1.1 protocol, set the environment variable GSK_PROTOCOL_TLSV1_1 to "ON". Similarly, to enable TLS V1.2 protocol, set the environment variable GSK_PROTOCOL_TLSV1_2 to "ON".

A SAF key ring name is specified as *userid/keyring*. The current user ID can be used if the user ID is omitted. The user must have READ access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by the current user. The user must have UPDATE access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by another user. Note certificate private keys are not available when using a SAF key ring owned by another user.

A PKCS #11 token is specified in the following format:

`*TOKEN*/`*`NAME`*

where *NAME* is the name of the PKCS #11 token. The user must have READ access to the `SO.`*`NAME`* and `USER.`*`NAME`* resources in the CRYPTOZ class when using a PKCS #11 token.

For information about System SSL, see *z/OS Cryptographic Services System SSL Programming*.

Function return value

The function return value is `LDAP_SUCCESS` if no error is detected. Otherwise, it is one of the LDAP error codes listed in the `ldap.h` include file.

The following are some common errors for this routine:

LDAP_PARAM_ERROR
: A parameter is not correct.

LDAP_SSL_ALREADY_INITIALIZED
: The SSL client run time is already initialized.

LDAP_SSL_INITIALIZE_FAILED
: SSL initialization failed.

LDAP_SSL_NOT_AVAILABLE
: System SSL is not available.

ldap_ssl_init()

Purpose

Create and initialize an LDAP handle for an SSL connection

Format

```
#include <ldap.h>

LDAP * ldap_ssl_init(
    const char *         host,
    int                  port,
    const char *         label)
```

Parameters

Input

host
Specifies the location of the LDAP server as a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable. This location can be a blank-separated host list or a single LDAP URL. Specify NULL for this parameter to connect to an LDAP server on the local system using the IPv4 loopback address (127.0.0.1).

port
Specifies the port for the LDAP server. This value can be used when an explicit port is not specified in the host list, and it must be between 1 and 65535. If 0 is specified, the default LDAP port (389) can be used.

label
Specifies the label for the client certificate as a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable. Specify NULL for this parameter to use the GSK_KEY_LABEL environment variable. If NULL is specified for this parameter and the GSK_KEY_LABEL environment variable is not defined, the default certificate for the SSL key database, SAF key ring, or PKCS #11 token can be used. A client certificate is needed only when the LDAP server is configured for client authentication.

Usage

The **ldap_ssl_init()** routine creates and initializes an LDAP handle. The routine does not establish a connection with the LDAP server. A connection is established when the first server request using the handle is issued. The handle is always initialized for an SSL connection even if an LDAP URL is specified for the host parameter and the URL scheme is **ldap** instead of **ldaps**. The application should call the **ldap_unbind()** or **ldap_unbind_s()** routine to release the handle when it is no longer needed. The location of the LDAP server can be explicitly specified by using a host list or an LDAP URL containing a host name. The location of the LDAP server can be implicitly specified by using an LDAP URL that does not contain a host name.

A host list consists of one or more blank-separated *host:port* values. The host specification is a DNS resource name (for example, dcesec4.endicott.ibm.com), a dotted decimal IPv4 address (for example, 9.130.25.34), or a colon-separated IPv6 address enclosed in square brackets (for example, [1080::8:800:200C:417A]. The port, if specified, must be a decimal number between 1 and 65535. The value of the

port parameter can be used if a port is not specified. The hosts are tried in the order specified until a connection is established with an LDAP server.

An LDAP URL has the following format:

[**<**][**URL:**]*scheme*://[*host*[:*port*]][/*dn*[?*attributes*[?*scope*[?*filter*]]]][**>**]

where:

scheme
: Specifies the value **ldap** for a non-SSL connection and **ldaps** for an SSL connection. However, the **ldap_ssl_init()** routine always sets up an SSL connection. Use the **ldap_init()** routine if you want the connection type to be determined by the URL scheme.

host:port
: Specifies the location of the LDAP server. The host specification can be a DNS resource name (for example, `dcesec4.endicott.ibm.com`), a dotted decimal IPv4 address (for example, `9.130.25.34`), or a colon-separated IPv6 address enclosed in square brackets (for example, `[1080::8:800:200C:417A]`). The port, if specified, must be a decimal number between `1` and `65535`. The port defaults to `389` for a non-SSL connection and `636` for an SSL connection.

dn
: Specifies the distinguished name (DN) for the request. The DN can be used as a filter when the **ldap_server_locate()** routine should be called to locate the LDAP server.

attributes
: Consists of one or more comma-separated search attributes. This value is not used by the **ldap_ssl_init()** routine.

scope
: Specifies the search scope and can be `"base"`, `"one"`, or `"sub"`. This value is not used by the **ldap_ssl_init()** routine.

filter
: Specifies the search filter. This value is not used by the **ldap_ssl_init()** routine.

The URL can be optionally enclosed in angle brackets or prefixed with `URL:` or both.

The **ldap_ssl_init()** routine calls the **ldap_server_locate()** routine to locate the LDAP server when the LDAP URL does not contain a host name. The default server information file `/etc/ldap/ldap_server_info.conf` can be used unless the `LDAP_SERVER_INFO_CONF` environment variable is defined. The **ldap_server_locate()** routine uses the default values for everything except the DN filter. The DN filter is set to the DN specified in the URL. (No DN filtering is done if a DN is not specified in the URL). A server entry is selected only if the security type is `LDAP_LSI_SSL`. A server entry is not selected if the security type is not defined.

The **ldap_ssl_client_init()** routine must be called before the **ldap_ssl_init()** routine.

The LDAP handle is initialized with the following default values. The **ldap_set_option()** or **ldap_set_option_np()** routine can be called to set different values upon completion of the **ldap_ssl_init()** routine.

- The LDAP protocol version is set based on the `LDAP_VERSION` environment variable. The protocol version is set to `3` if the `LDAP_VERSION` environment variable is not defined.

- The LDAP version 2 wire format is set based on the LDAP_V2_WIRE_FORMAT environment variable. The LDAP version 2 wire format is set to UTF-8 if the LDAP_V2_WIRE_FORMAT environment variable is not defined.
- Referral processing is enabled and the referral hop limit is set to 10.

Function return value

The function return value is the new LDAP handle if no error is detected. Otherwise, the return value is NULL.

ldap_start_tls_s_np()

Purpose

Start TLS for a connection

Format

```
#include <ldap.h>

int ldap_start_tls_s_np(
    LDAP *                  ld,
    const char *            label)
```

Parameters

Input

ld Specifies the LDAP handle.

label
Specifies the label for the client certificate as a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. Specify NULL for this parameter to use the GSK_KEY_LABEL environment variable. If you specify NULL for this parameter and the GSK_KEY_LABEL environment variable is not defined, the default certificate for the SSL key database, SAF key ring, or PKCS #11 token can be used. A client certificate is needed only when the LDAP server is configured for client authentication.

Usage

The **ldap_start_tls_s_np()** routine initiates Transport Layer Security (TLS) for an existing connection with an LDAP server. An error is returned if TLS is already being used by the connection or if there are outstanding requests. Any existing authentication for the connection remains unchanged. If the application wants to use the client certificate for authentication, it should call the **ldap_sasl_bind()** or **ldap_sasl_bind_s()** routine after calling **ldap_start_tls_s_np()** and specify EXTERNAL as the SASL authentication method.

The **ldap_ssl_client_init()** routine must be called to initialize the SSL environment before calling the **ldap_start_tls_s_np()** routine.

The certificate presented by the LDAP server must contain the DNS host name as either the common name (CN) portion of the certificate subject name or as a subject alternate name. The DNS host name is the name specified when the **ldap_init()** or **ldap_ssl_init()** routine was called. An error is returned if the server certificate does not contain a matching host name.

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes listed in the ldap.h include file.

The following are some common errors for this routine:

LDAP_INAPPROPRIATE_AUTH
The server certificate does not contain the DNS host name for the connection.

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_OPERATIONS_ERROR
TLS is already active or there are outstanding requests for the connection.

LDAP_PARAM_ERROR
A parameter is not valid.

LDAP_PROTOCOL_ERROR
The **Start TLS** extended operation is not supported.

LDAP_SSL_CLIENT_INIT_NOT_CALLED
The **ldap_ssl_client_init()** routine has not been called.

LDAP_UNAVAILABLE
TLS support is not available or the server is stopping.

ldap_stop_tls_s_np()

Purpose

Stop TLS for a connection

Format

```
#include <ldap.h>

int ldap_stop_tls_s_np(
    LDAP *                  ld)
```

Parameters

Input

ld Specifies the LDAP handle.

Usage

The **ldap_stop_tls_s_np()** routine stops Transport Layer Security (TLS) for a connection. The routine returns an error if TLS is not being used or if there are outstanding requests. The connection reverts to anonymous authentication.

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes listed in the ldap.h include file.

The following are some common errors for this routine:

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_OPERATIONS_ERROR
TLS is not active or there are outstanding requests for the connection.

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_unbind(), ldap_unbind_s()

Purpose

Close the connection to the LDAP server and release the LDAP handle

Format

```
#include <ldap.h>

int ldap_unbind(
    LDAP *                ld)

int ldap_unbind_s(
    LDAP *                ld)
```

Parameters

Input

ld Specifies the LDAP handle.

Usage

The **ldap_unbind()** or **ldap_unbind_s()** routine closes the connection to the LDAP server and releases the LDAP handle. The LDAP handle cannot be used upon completion of either routine. Control is not returned to the application until the LDAP handle is released. (Both routines are synchronous.)

ldap_unbind() and **ldap_unbind_s()** return an error if the routine should be called while an application exit routine is active for the LDAP handle.

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes listed in the ldap.h include file.

The following are some common errors for this routine:

LDAP_INVALID_STATE
Unbind already started for the LDAP handle or an application exit is active.

LDAP_PARAM_ERROR
The LDAP handle is not valid.

LDAP_SERVER_DOWN
Unable to send unbind request to server.

ldap_url_parse()

Purpose

Parse an LDAP URL

Format

```
#include <ldap.h>

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

Parameters

Input

url
Specifies the LDAP URL as a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable.

Output

ludpp
Returns the address of the LDAP URL description. The application should call the **ldap_free_urldesc()** routine to release the URL description when it is no longer needed. Text data is returned in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable.

Usage

The **ldap_url_parse()** routine parses an LDAP URL and returns an LDAP URL description.

An LDAP URL has the following format:

[**<**][**URL:**]*scheme*[**://**][*host*[**:***port*]][**/***dn*[**?***attributes*[**?***scope*[**?***filter*]]]][**>**]

where:

scheme
Specifies the value **ldap** for a non-SSL connection and **ldaps** for an SSL connection.

host:port
Specifies the location of the LDAP server. The host specification can be a DNS resource name (for example, dcesec4.endicott.ibm.com), a dotted decimal IPv4 address (for example, 9.130.25.34), or a colon-separated IPv6 address enclosed in square brackets (for example, [1080::8:800:200C:417A]). The port, if specified, must be a decimal number between 1 and 65535. The port defaults to 389 for a non-SSL connection and 636 for an SSL connection.

dn Specifies the distinguished name (DN) for the request.

attributes
Consists of one or more comma-separated search attributes.

scope Specifies the search scope and can be "base", "one", or "sub".

filter Specifies the search filter. The filter is set to "(objectClass=*)" if no search filter is specified.

The URL can be optionally enclosed in angle brackets or prefixed with URL: or both.

A URL consists of characters in the US-ASCII character set (the characters from the ISO8859-1 code page with values between 1 and 127). Escaped characters can be specified in the scheme-specific section. Escaped characters are used for characters not in the US-ASCII character set or for characters that are reserved for control purposes (such as ?) or which might cause problems depending on how the URL can be used (such as embedded blanks). An escaped character consists of a percent (%) followed by two hexadecimal digits representing the character value in the ISO8859-1 code page. For example, a blank is represented at %20. For more information, see RFC 1738: *Uniform Resource Locators (URL)*.

The LDAPURLDesc structure is defined as follows:

```
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;
    unsigned long       lud_options;
} LDAPURLDesc;
```

where:

lud_host
Returns the LDAP server host name as a null-terminated character string in UTF-8 or the local EBCDIC code page depending on the LDAP_LIBASCII compiler variable. If the URL does not specify a host name, this field is set to NULL.

lud_port
Returns the LDAP server port number. If the URL does not specify a port number, the port number is set to 389 for a non-SSL connection or 636 for an SSL connection.

lud_dn Returns the distinguished name as a null-terminated character string in UTF-8 or the local EBCDIC code page depending on the LDAP_LIBASCII compiler variable. If the URL does not specify a distinguished name, this field is set to NULL.

lud_attrs
Returns an array of search attributes where each attribute is a null-terminated character string in UTF-8 or the local EBCDIC code page depending on the LDAP_LIBASCII compiler variable. The array is terminated by a NULL address. If the URL does not specify any search attributes, this field is set to NULL.

lud_scope
Returns the search scope and is set to LDAP_SCOPE_BASE, LDAP_SCOPE_ONELEVEL, or LDAP_SCOPE_SUBTREE. If the URL does not specify a search scope, the scope is set to LDAP_SCOPE_BASE.

lud_filter
Returns the search filter as a null-terminated character string in UTF-8 or

the local EBCDIC code page depending on the LDAP_LIBASCII compiler variable. If the URL does not specify a search filter, this field is set to the string "(objectClass=*)".

lud_string
Returns a copy of the original URL as a null-terminated character string in UTF-8 or the local EBCDIC code page depending on the LDAP_LIBASCII compiler variable.

lud_options
Specifies the LDAP_URL_OPT_SECURE flag, which is set when the URL specifies an SSL connection.

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes listed in the ldap.h include file.

The following are some common errors for this routine:

LDAP_PARAM_ERROR
URL address is NULL.

LDAP_URL_ERR_BADPORT
Server port is not valid.

LDAP_URL_ERR_BADSCOPE
Search scope is not valid.

LDAP_URL_ERR_MALFORMED
URL syntax is not valid.

LDAP_URL_ERR_MEM
Insufficient storage is available.

LDAP_URL_ERR_NOTLDAP
URL does not specify an LDAP scheme.

ldap_url_parse_np()

Purpose

Parse an LDAP URL

Format

```
#include <ldap.h>

int ldap_url_parse_np(
    LDAP *                  ld,
    const char *            url,
    LDAPURLDesc **          ludpp)
```

Parameters

Input

ld Specifies an LDAP handle. This parameter can be specified as NULL if the URL is in UTF-8. Otherwise, the URL is in either the local EBCDIC code page or UTF-8, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle.

url
Specifies the LDAP URL as a null-terminated character string in either the local EBCDIC code page or UTF-8, as determined by the LDAP handle.

Output

ludpp
Returns the address of the LDAP URL description. The application should call the **ldap_free_urldesc()** routine to release the URL description when it is no longer needed.

Usage

The **ldap_url_parse_np()** routine is the same as the **ldap_url_parse()** routine except that text strings provided by the application and text strings returned to the application are in either UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option. For more information about the **ldap_url_parse()** routine see "ldap_url_parse()" on page 207.

Function return value

The function return value is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes listed in the ldap.h include file.

The following are some common errors for this routine:

LDAP_PARAM_ERROR
URL address is NULL.

LDAP_URL_ERR_BADPORT
Server port is not valid.

LDAP_URL_ERR_BADSCOPE
Search scope is not valid.

LDAP_URL_ERR_MALFORMED
URL syntax is not valid.

LDAP_URL_ERR_MEM
Insufficient storage is available.

LDAP_URL_ERR_NOTLDAP
URL does not specify an LDAP scheme.

ldap_url_search(), ldap_url_search_s(), ldap_url_search_st()

Purpose

Search the LDAP directory using an LDAP URL

Format

```
#include <ldap.h>

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

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

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

Parameters

Input

ld Specifies the LDAP handle.

url
Specifies the LDAP URL as a null-terminated character string in either the local EBCDIC code page or UTF-8, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle.

attrsonly
Specifies whether the attribute values should be returned along with the attribute types. A nonzero value causes just the attribute types to be returned. A zero value causes both attribute types and attribute values to be returned.

timeout
Specifies the maximum time for the search request. Specify NULL for this parameter if there is no time limit for the request. Otherwise, set the tv_sec field to the maximum time in seconds. Note that the actual time limit is the smaller of the client-specified value and the maximum time allowed by the LDAP server.

Output

result
Returns the address of the result message chain. If the LDAP server returns no result messages, the message address is set to NULL. Note that the synchronous routines can return one or more result messages even when the function return value is not LDAP_SUCCESS. The application should call the **ldap_msgfree()** routine to release the message chain when it is no longer needed.

Usage

The **ldap_url_search()** routine initiates the search and returns control to the application. The application must call the **ldap_result()** routine to obtain the search results.

The **ldap_url_search_s()** and **ldap_url_search_st()** routines initiate the search and wait for the search results. The **ldap_url_search_s()** routine waits indefinitely, while the **ldap_url_search_st()** routine provides a parameter to specify a time limit. The search request is abandoned if the client is unable to wait for the response because of an error from the **ldap_result()** routine. The search request is also abandoned if the time limit specified for the **ldap_url_search_st()** routine expires.

The **ldap_url_search()**, **ldap_url_search_s()** and **ldap_url_search_st()** routines are like the **ldap_search()**, **ldap_search_s()** and **ldap_search_st()** routines. The base object name, search scope, search filter, and attribute list are obtained from the LDAP URL. For more information about searching, see the description of the search routines in "ldap_search(), ldap_search_s(), ldap_search_st(), ldap_search_ext(), ldap_search_ext_s()" on page 164.

An LDAP URL has the following format:

```
[<][URL:]scheme://[host[:port]][/dn[?attributes[?scope[?filter]]]][>]
```

where:

scheme
: Specifies the value **ldap** for a non-SSL connection and **ldaps** for an SSL connection.

host:port
: Specifies the location of the LDAP server. The host specification can be a DNS resource name (for example, `dcesec4.endicott.ibm.com`), a dotted decimal IPv4 address (for example, `9.130.25.34`), or a colon-separated IPv6 address that is enclosed in square brackets (for example, `[1080::8:800:200C:417A]`). The port, if specified, must be a decimal number between `1` and `65535`. The port defaults to `389` for a non-SSL connection and `636` for an SSL connection.

dn
: Specifies the base object name for the search request.

attributes
: Consists of one or more comma-separated search attributes. All attributes are returned if no search attributes are specified.

scope
: Specifies the search scope and can be `"base"`, `"one"`, or `"sub"`. The scope is set to `"base"` if no search scope is specified.

filter
: Specifies the search filter. The filter is set to `"(objectClass=*)"` if no search filter is specified.

The URL can be optionally enclosed in angle brackets or prefixed with `URL:` or both.

A URL consists of characters in the US-ASCII character set (the characters from the ISO8859-1 code page with values between 1 and 127). Escaped characters can be specified in the scheme-specific section. Escaped characters are used for characters not in the US-ASCII character set or for characters that are reserved for control purposes (such as `?`) or that might cause problems depending on how the URL can be used (such as embedded blanks). An escaped character consists of a percent (`%`)

followed by two hexadecimal digits representing the character value in the ISO8859-1 code page. For example, a blank is represented at %20. For more information, see RFC 1738: *Uniform Resource Locators (URL)*.

The URL must specify a host name. The port defaults to 389 for a non-SSL connection and 636 for an SSL connection. The existing connection can be used if the security type, host name, and port in the LDAP URL are the same as the values used to establish the connection. Otherwise, a new connection is established for the search request. The application rebind procedure is started if a new connection is established to obtain the bind parameters. An unauthenticated connection is established if the application did not provide a rebind procedure.

The **ldap_ssl_client_init()** routine must have been called to initialize the SSL environment if the URL specifies an SSL connection. The certificate label specified for the **ldap_ssl_init()** or **ldap_start_tls_s_np()** routine can be used for the client certificate. If no label has been set, the label specified by the GSK_KEY_LABEL environment variable can be used. If no label has been set and the GSK_KEY_LABEL environment variable is not defined, the default certificate for the SSL key database or SAF key ring can be used. Note a client certificate is needed only when the LDAP server is configured for client authentication.

Function return value

The **ldap_url_search()** routine returns -1 if a client error is detected. Otherwise, it returns the message identifier assigned to the search request. The application should call the **ldap_get_errno()** routine to get the error code if the return value is -1. Errors reported by the LDAP server are not returned by the **ldap_url_search()** routine. The application must call the **ldap_parse_result()** routine to obtain the result code from the search done message returned by the **ldap_result()** routine.

The **ldap_url_search_s()** and **ldap_url_search_st()** routines return LDAP_SUCCESS if the request is successful. Otherwise, the return value is one of the error codes listed in the ldap.h include file. The return value includes errors detected by the LDAP client and errors detected by the LDAP server. One or more result messages can be returned by these routines even when the return value is not LDAP_SUCCESS. If no result messages are returned, the result message address is NULL.

The following are some common client errors:

LDAP_FILTER_ERROR
The search filter is not valid.

LDAP_INVALID_STATE
An unbind request has been issued for the LDAP handle.

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_NOT_SUPPORTED
The LDAP protocol version must be LDAP_VERSION3 to use an extensible filter item.

LDAP_PARAM_ERROR
A parameter is not valid.

LDAP_SERVER_DOWN
Network connection failed.

LDAP_TIMEOUT
The wait time has expired and the search request has been abandoned.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION
A critical client control is either not recognized or is not supported for a search operation.

LDAP_URL_ERR_BADPORT
Server port is not valid.

LDAP_URL_ERR_BADSCOPE
Search scope is not valid.

LDAP_URL_ERR_MALFORMED
URL syntax is not valid.

LDAP_URL_ERR_MEM
Insufficient storage is available.

LDAP_URL_ERR_NOTLDAP
URL does not specify an LDAP scheme.

The following are some common search result codes:

LDAP_INSUFFICIENT_ACCESS
Not authorized to access base object.

LDAP_NO_SUCH_OBJECT
The base object is not found.

LDAP_REFERRAL
The base object is not in the current LDAP server.

LDAP_SIZELIMIT_EXCEEDED
The search size limit has been exceeded.

LDAP_TIMELIMIT_EXCEEDED
The search time limit has been exceeded.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION
A critical server control is either not recognized or is not supported for a search operation.

ldap_value_free()

Purpose

Release storage allocated for an array of character strings

Format

```
#include <ldap.h>

void ldap_value_free(
    char *                vals[])
```

Parameters

Input

vals
Specifies the array of character strings. The end of the array is indicated by a NULL address.

Usage

The **ldap_value_free()** routine releases the storage allocated for an array of character strings.

Function return value

There is no function return value.

ldap_value_free_len()

Purpose

Release storage allocated for an array of binary values

Format

```
#include <ldap.h>

void ldap_value_free_len(
    BerVal *             vals[])
```

Parameters

Input

vals
Specifies the array of binary values. The end of the array is indicated by a NULL address.

Usage

The **ldap_value_free_len()** routine releases the storage allocated for an array of binary values.

Function return value

There is no function return value.

ldap_version()

Purpose

Return LDAP version information

Format

```
#include <ldap.h>

int ldap_version(
    LDAPVersion *          info)
```

Parameters

Output

info
Returns the LDAP version information in the LDAPVersion structure provided by the application. NULL can be specified for this parameter if the version information is not needed.

Usage

The **ldap_version()** routine returns information about the LDAP runtime library.

The following fields are set in the LDAPVersion structure:

```
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;
```

where:

sdk_version
This field is set to the LDAP run time library version and release *(vrr)*. The version value (*v*) is 4, indicating z/OS Version 2. The release value (*rr*) is 10 + the z/OS release number. For example, z/OS V2R1, the *sdk_version* is 411.

protocol_version
This field is set to the highest LDAP protocol version (multiplied by 100) supported by the LDAP runtime library. This is currently 300 for LDAP Version 3.

SSL_version
This field is set to the highest SSL protocol version (multiplied by 100) supported by the LDAP runtime library.

security_level
This field is not used and is set to 0.

ssl_max_cipher
This field is set to the SSL Version 3/TLS Version 1 cipher suites recognized by LDAP. This is the same as the LDAP_SSL_CIPHERLIST definition in the ldap.h include file. The returned values are in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII

compiler variable. The LDAP_OPT_SSL_CIPHER option of the **ldap_get_option()** routine can be used after the SSL environment is initialized to obtain the actual cipher suite list. This field is deprecated.

Restriction: The actual cipher suites available on a given system are determined by the System SSL product and can be affected by government export regulations and the values for various System SSL environment variables.

ssl_min_cipher
This field is set to the same value as the *ssl_max_cipher* field. This field is deprecated.

For more information about System SSL, see *z/OS Cryptographic Services System SSL Programming*.

Function return value

The return value is the LDAP runtime version and release value. This is the value returned in the *sdk_version* field of the LDAPVersion structure.

Chapter 3. Deprecated LDAP routines

This topic describes the deprecated LDAP routines. These routines have been replaced by newer routines, but are still supported.

Guideline: If you are writing new applications, use the routines that are described in Chapter 2, "LDAP routines," on page 23 instead of the deprecated routines. If you are updating existing applications that use deprecated LDAP routines, consider updating them to use the newer routines. They are listed in each section of this topic as *preferred* routines.

ldap_bind(), ldap_bind_s()

Preferred routines

ldap_simple_bind() or **ldap_simple_bind_s()**

Purpose

Bind to the LDAP server using a distinguished name (DN) and password

Format

```
#include <ldap.h>

int ldap_bind (
    LDAP *              ld,
    const char *        who,
    const char *        passwd,
    int                 method)

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

Parameters

Input

ld Specifies the LDAP handle.

who
Specifies the distinguished name as a null-terminated character string. The distinguished name is in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. An anonymous bind is performed if this parameter is NULL or the distinguished name is a zero-length string.

passwd
Specifies the password as a null-terminated character string. The password is in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle.

method
Specifies the bind method and must be LDAP_AUTH_SIMPLE.

Usage

The **ldap_bind()** or **ldap_bind_s()** routine binds to the LDAP server identified by the LDAP handle. The LDAP server authenticates the client using the distinguished name and password. Note that this information is sent unencrypted to the LDAP server unless an SSL connection is used.

The **ldap_bind()** routine sends the bind message to the LDAP server and returns control to the application. The application should call the **ldap_result()** routine to get the response to the bind request.

The **ldap_bind_s()** routine sends the bind message to the LDAP server and waits for a response. The bind request is abandoned if the client is unable to wait for the response due to an error from the **ldap_result()** routine.

The **ldap_bind()** and **ldap_bind_s()** routines use client controls specified by the LDAP_OPT_CLIENT_CONTROLS and server controls specified by the LDAP_OPT_SERVER_CONTROLS options.

Function return value

The function return value for the **ldap_bind()** routine is the message identifier of the bind message, or -1 if a client error occurred. When the return value is -1, the application should call the **ldap_get_errno()** routine to get the LDAP error code. The **ldap_bind()** routine does not return any errors reported by the LDAP server. The application must call the **ldap_parse_result()** routine to obtain the error code from the bind response message returned by the **ldap_result()** routine.

The function return value for the **ldap_bind_s()** routine is LDAP_SUCCESS if no error is detected. Otherwise, it is one of the LDAP error codes listed in the ldap.h include file. The **ldap_bind_s()** routine returns errors reported by the LDAP server and errors detected by the LDAP client.

The following are some common client errors:

LDAP_AUTH_UNKNOWN
Method is not LDAP_AUTH_SIMPLE.

LDAP_INVALID_STATE
A bind or unbind is in progress for the LDAP handle or an application exit is active for the LDAP handle.

LDAP_LOCAL_ERROR
A system function reported an error.

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_PARAM_ERROR
A parameter is not valid.

LDAP_SERVER_DOWN
Unable to connect to the LDAP server.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION
A critical client control is either not recognized or is not supported for a bind operation.

The following are some common bind result codes:

LDAP_INAPPROPRIATE_AUTH
The client provided inappropriate authentication.

LDAP_INVALID_CREDENTIALS
The credentials provided by the client are not valid.

LDAP_REFERRAL
The server cannot accept the bind.

LDAP_STRONG_AUTH_REQUIRED
The server requires strong authentication.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION
A critical server control is either not recognized or is not supported for a bind operation.

ldap_modrdn(), ldap_modrdn_s()

Preferred routines

ldap_rename() or **ldap_rename_s()**

Purpose

Rename an entry in the LDAP directory

Format

```
#include <ldap.h>

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

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

Parameters

Input

ld Specifies the LDAP handle.

dn Specifies the distinguished name for the directory entry as a null-terminated character string in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle. A zero-length name is not allowed for a rename request.

newrdn
Specifies the new relative distinguished name (RDN) for the directory entry as a null-terminated character string in UTF-8 or the local EBCDIC code page, as determined by the LDAP_OPT_UTF8_IO option for the LDAP handle.

deleteoldrdn
Specify TRUE if the attributes from the old RDN are to be removed from the entry. Specify FALSE if the attributes are to be retained.

Usage

The **ldap_modrdn()** routine sends the request to the LDAP server and returns control to the application. The application must call the **ldap_result()** routine to obtain the result.

The **ldap_modrdn_s()** routine sends the request to the LDAP server and waits for the completion of the request. The modify request is abandoned if the client is unable to wait for the response because of an error from the **ldap_result()** routine.

The RDN for the requested directory entry is changed. The entry might or might not have subordinate entries. If the entry is not a leaf entry, the entire subtree is renamed.

The **ldap_modrdn()** and **ldap_modrdn_s()** routines use client controls specified by the LDAP_OPT_CLIENT_CONTROLS option and server controls specified by the LDAP_OPT_SERVER_CONTROLS option.

Function return value

The **ldap_modrdn()** routine returns -1 if a client error is detected. Otherwise, it returns the message identifier assigned to the rename request. If the return value is -1, the application should call the **ldap_get_errno()** routine to get the error code. The **ldap_modrdn()** routine does not return errors reported by the LDAP server. The application must call the **ldap_parse_result()** routine to obtain the result code from the result message returned by the **ldap_result()** routine.

The **ldap_modrdn_s()** routine returns LDAP_SUCCESS if the request is successful. Otherwise, the return value is one of the error codes listed in the ldap.h include file. The return value includes errors detected by the LDAP client and errors detected by the LDAP server.

The following are some common client errors:

LDAP_INVALID_STATE
An unbind request has been issued for the LDAP handle.

LDAP_NO_MEMORY
Insufficient storage is available.

LDAP_NOT_SUPPORTED
The LDAP protocol version must be LDAP_VERSION3 to specify server or client controls.

LDAP_PARAM_ERROR
A parameter is not valid.

LDAP_SERVER_DOWN
Network connection failed.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION
A critical client control is either not recognized or is not supported for a rename operation.

The following are some common server result codes:

LDAP_ALREADY_EXISTS
An entry with the new name exists.

LDAP_INSUFFICIENT_ACCESS
Not authorized to modify the directory entry.

LDAP_NO_SUCH_OBJECT
The directory entry does not exist.

LDAP_REFERRAL
The entry is not in the current LDAP server.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION
A critical server control is either not recognized or is not supported for a rename operation.

ldap_open()

Preferred routines

ldap_init() or **ldap_ssl_init()**

Purpose

Create and initialize an LDAP handle and then connect to the LDAP server

Format

```
#include <ldap.h>

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

Parameters

Input

host
Specifies the location of the LDAP server as a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable. This location can be a blank-separated host list or a single LDAP URL. Specify NULL for this parameter to connect to an LDAP server on the local system using the IPv4 loopback address (127.0.0.1).

port
Specifies the port for the LDAP server. This port is used when the host list does not specify an explicit port. The value must be between 1 and 65535. If you specify 0, the default LDAP port (389) is used.

Usage

The **ldap_open()** routine creates and initializes an LDAP handle and connects to the LDAP server. The handle is initialized for a non-SSL connection unless an LDAP URL is specified for the host parameter and the URL scheme is **ldaps** instead of **ldap**. The application should call the **ldap_unbind()** or **ldap_unbind_s()** routine to release the handle when it is no longer needed. The location of the LDAP server can be explicitly specified by using a host list or an LDAP URL containing a host name. The location of the LDAP server can be implicitly specified by using an LDAP URL that does not contain a host name.

A host list consists of one or more blank-separated *host:port* values. The *host* specification is a DNS resource name (for example, dcesec4.endicott.ibm.com), a dotted decimal IPv4 address (for example, 9.130.25.34), or a colon-separated IPv6 address that is enclosed in square brackets (for example, [1080::8:800:200C:417A]. The *port* specification is a decimal number between 1 and 65535. If a port is not specified, the value of the port parameter is used. The hosts are tried in the order that is specified until a connection is established with an LDAP server.

An LDAP URL has the following format:

[**<**][**URL:**]*scheme*://[*host*[**:***port*]][**/***dn*[**?***attributes*[**?***scope*[**?***filter*]]]][**>**]

where:

scheme Specifies the value **ldap** for a non-SSL connection and **ldaps** for an SSL connection.

host:port
Specifies the location of the LDAP server. The host specification can be a DNS resource name (for example, dcesec4.endicott.ibm.com), a dotted decimal IPv4 address (for example, 9.130.25.34), or a colon-separated IPv6 address that is enclosed in square brackets (for example, [1080::8:800:200C:417A]). The port, if specified, must be a decimal number between 1 and 65535. The port defaults to 389 for a non-SSL connection and 636 for an SSL connection.

dn Specifies the distinguished name (DN) for the request. The DN is used as a filter when the **ldap_server_locate()** routine is called to locate the LDAP server.

attributes
Consists of one or more comma-separated search attributes. This value is not used by the **ldap_open()** routine.

scope Specifies the search scope and can be "base", "one", or "sub". This value is not used by the **ldap_open()** routine.

filter Specifies the search filter. This value is not used by the **ldap_open()** routine.

The URL can be optionally enclosed in angle brackets or prefixed with URL: or both.

The **ldap_open()** routine calls the **ldap_server_locate()** routine to locate the LDAP server when the LDAP URL does not contain a host name. The default server information file /etc/ldap/ldap_server_info.conf is used unless the LDAP_SERVER_INFO_CONF environment variable is defined. The **ldap_server_locate()** routine uses the default values for everything except the DN filter. The DN filter is set to the DN specified in the URL (no DN filtering is done if a DN is not specified in the URL). The scheme specified in the URL is used to select servers from the list returned by the **ldap_server_locate()** routine. A server entry is selected if the scheme is **ldap** and the security type is LDAP_LSI_NOSSL or if the scheme is **ldaps** and the security type is LDAP_LSI_SSL. A server entry is not selected if the security type is not defined.

The **ldap_ssl_client_init()** routine must be called before the **ldap_open()** routine if the LDAP URL specifies an SSL connection.

The LDAP handle is initialized with the following default values. The **ldap_set_option()** or **ldap_set_option_np()** routine can be called to set different values upon completion of the **ldap_open()** routine.

- The LDAP protocol version is set based on the LDAP_VERSION environment variable. If the LDAP_VERSION environment variable is not defined the protocol version is set to 2.
- The LDAP version 2 wire format is set based on the LDAP_V2_WIRE_FORMAT environment variable. If the LDAP_V2_WIRE_FORMAT environment variable is not defined the LDAP version 2 wire format is set to UTF-8.
- Referral processing is enabled and the referral hop limit is set to 10.

Function return value

The function return value is the new LDAP handle if no error is detected. Otherwise, the return value is `NULL`.

ldap_perror()

Preferred routines

ldap_parse_result() or **ldap_get_errno()**

Purpose

Print an error message on `stderr`

Format

```
#include <ldap.h>

void ldap_perror (
    LDAP *               ld,
    const char *         prefix)
```

Parameters

Input

ld Specifies the LDAP handle.

prefix
Specifies the message prefix as a null-terminated character string in either the local EBCDIC code page or UTF-8, as determined by the LDAP handle. If `NULL` is specified for this parameter, a message prefix is not used.

Usage

The **ldap_perror()** routine prints an error message on `stderr`. The printed text is in the local EBCDIC code page or UTF-8, as determined by the `LDAP_LIBASCII` compiler variable.

The last error associated with the LDAP handle is used to retrieve the error text for the error. The first line is printed as *prefix*`:`*text*.

If **ldap_result2error()** or one of the synchronous request routines is called before the **ldap_perror()** routine, two additional lines are printed. If the result message contained a value for the matched distinguished name, this value is printed as *prefix*`:matched:`*name*. If the result message contained an error message, this value is printed as *prefix*`:additional info:`*message*. The **ldap_perror()** routine continues to print the same values for matched distinguished name and error message on subsequent calls until new values are set by **ldap_result2error()** or one of the synchronous search request routines.

Function return value

There is no function return value.

ldap_result2error()

Preferred routines

ldap_parse_result()

Purpose

Return the error code for an LDAP result message

Format

```
#include <ldap.h>

int ldap_result2error (
    LDAP *                ld,
    LDAPMessage *         result,
    int                   freeit)
```

Parameters

Input

ld Specifies the LDAP handle.

result
Specifies the result message returned by **ldap_result()** or one of the synchronous request routines.

freeit
Specify TRUE to free the LDAP message chain before returning to the application. Specify FALSE to keep the LDAP message chain. If you specify TRUE, the message chain is freed even when the function return value is not LDAP_SUCCESS.

Usage

The **ldap_result2error()** routine returns the error code from the LDAP result message. An error is returned if **ldap_result2error()** is called for a search entry or search reference message and the message chain does not contain the search result message (the message chain is still released if the *freeit* parameter is nonzero).

Function return value

The function return value is the result code from the LDAP result message. In addition, the following error codes can be returned if an error is detected by the **ldap_result2error()** routine:

LDAP_NO_RESULT_MESSAGE
The message chain does not contain an LDAP result.

LDAP_PARAM_ERROR
A parameter is not valid.

ldap_ssl_start()

Preferred routines

ldap_ssl_client_init() and **ldap_ssl_init()**

Purpose

Start an SSL connection with the LDAP server

Format

```
#include <ldap.h>

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

Parameters

Input

ld LDAP handle created by the **ldap_open()** routine. An error is returned if the handle is created by the **ldap_init()** or **ldap_ssl_init()** routine.

keyring
Specifies the name of the SSL key database, SAF key ring, or PKCS #11 token as a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable. Specify NULL for this parameter to use the GSK_KEYRING_FILE environment variable. An SSL key database must be a z/OS UNIX System Services file and cannot be a partitioned or sequential data set. Specify a zero-length character string to use a SAF key ring or PKCS #11 token instead of a key database.

keyring_pw
Specifies the password for the SSL key database as a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable. Specify **file://***filename* to use an SSL stash file, where *filename* is the name of the stash file. An SSL stash file must be a file system file and cannot be a partitioned or sequential data set. Specify a zero-length character string to use a SAF key ring instead of a key database. Specify NULL for this parameter to use the GSK_KEYRING_PW or GSK_KEYRING_STASH environment variable. If you specify NULL and the GSK_KEYRING_PW and GSK_KEYRING_STASH environment variables are not defined, a SAF key ring is used. If you specify NULL for the keyring parameter, this parameter is ignored.

label
Specifies the label for the client certificate as a null-terminated character string in the local EBCDIC code page or UTF-8, as determined by the LDAP_LIBASCII compiler variable. Specify NULL for this parameter to use the GSK_KEY_LABEL environment variable. If you specify NULL for this parameter and the GSK_KEY_LABEL environment variable is not defined, the default certificate for the SSL key database, SAF key ring, or PKCS #11 token is used. A client certificate is needed only when the LDAP server is configured for client authentication.

Usage

The **ldap_ssl_start()** routine starts an SSL connection with the LDAP server. The LDAP handle must be created by the **ldap_open()** routine and not by the **ldap_init()** or **ldap_ssl_init()** routine. It is not necessary to call the **ldap_ssl_client_init()** routine because the **ldap_ssl_start()** routine initializes the SSL environment. The *keyring* and *keyring_pw* parameters are ignored if the SSL environment has already been initialized by a prior call to either the **ldap_ssl_client_init()** or **ldap_ssl_start()** routine. LDAP does not support SSL V2 protocol, and disables it from being used. SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 protocols are supported. The z/OS System SSL defaults and environment variables control which of these supported protocols are enabled or disabled. For example, the environment variable GSK_PROTOCOL_SSLV3 can be set to "ON" to enable SSL V3 protocol, or "OFF" to disable SSL V3 protocol. The environment variable GSK_PROTOCOL_TLSV1 can be set to "ON" to enable TLS V1.0 protocol, or "OFF" to disable TLS V1.0 protocol. TLS V1.1 and TLS V1.2 protocols are disabled by default. To enable TLS V1.1 protocol, set the environment variable GSK_PROTOCOL_TLSV1_1 to "ON". Similarly, to enable TLS V1.2 protocol, set the environment variable GSK_PROTOCOL_TLSV1_2 to "ON".

A SAF key ring name is specified as *userid/keyring*. The current user ID is used if *userid* is omitted. The user must have READ access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by the current user. The user must have UPDATE access to the IRR.DIGTCERT.LISTRING resource in the FACILITY class when using a SAF key ring owned by another user. Note certificate private keys are not available when using a SAF key ring owned by another user.

A PKCS #11 token is specified in the following format:

`*TOKEN*/`*`NAME`*

where *`NAME`* is the name of the PKCS #11 token. The user must have READ access to the `SO.`*`NAME`* and `USER.`*`NAME`* resources in the CRYPTOZ class when using a PKCS #11 token.

For more information about System SSL, see *z/OS Cryptographic Services System SSL Programming*.

Function return value

The function return value is `LDAP_SUCCESS` if no error is detected. Otherwise, it is one of the LDAP error codes listed in the `ldap.h` include file.

The following are some common errors for this routine:

LDAP_INVALID_STATE
: LDAP handle is in incorrect state.

LDAP_PARAM_ERROR
: A parameter is not correct.

LDAP_SSL_HANDSHAKE_FAILED
: The SSL handshake failed.

LDAP_SSL_INITIALIZE_FAILED
: SSL initialization failed.

LDAP_SSL_NOT_AVAILABLE
System SSL is not available.

LDAP_SSL_PARAM_ERROR
An SSL parameter is not correct.

Chapter 4. Using the LDAP client

LDAP client environment variables

The following environment variables are processed during LDAP client run time initialization when the first LDAP API routine is called. Changes to these environment variables after this time have no effect.

- LDAP_CLIENT_CACHE
- LDAP_CLIENT_CACHE_MAX_SIZE
- LDAP_CLIENT_CACHE_TTL
- LDAP_DEBUG
- LDAP_DEBUG_FILENAME
- LDAP_ERROR_LOGGING
- LDAP_STDERR_FILENAME
- LDAP_STDOUT_FILENAME

The following environment variables are processed as needed by the LDAP client run time. Changes to these environment variables take effect the next time they are used by the LDAP client run time, typically when a new LDAP handle is created.

- LDAP_EXC_ABEND_DUMP
- LDAP_SERVER_INFO_CONF
- LDAP_SSL_CIPHER_FORMAT
- LDAP_VERSION
- LDAP_V2_WIRE_FORMAT
- LOCALDOMAIN
- RESOLVER_CONFIG
- SOCKS_CONF
- SOCKS_PASSWORD
- SOCKS_SERVER
- SOCKS_USERNAME
- SOCKS_VERSION

Each environment variable is briefly described as follows:

LDAP_CLIENT_CACHE
: Controls global search result caching. Specify ON to enable global search result caching. Specify OFF to disable global search result caching. The default is no global search result caching. All LDAP handles use the global cache unless you use the **ldap_memcache_set()** routine to specify a different cache for an LDAP handle. All search results are cached when using the global search result cache.

LDAP_CLIENT_CACHE_MAX_SIZE
: Specifies the maximum size in bytes for the global search result cache. A value of 0 indicates that there is no maximum size. The default is 0 if the LDAP_CLIENT_CACHE_MAX_SIZE environment variable is not defined. Older entries are removed from the cache to make room for new entries when the maximum cache size is reached.

LDAP_CLIENT_CACHE_TTL
Specifies the maximum time in seconds that an entry is retained in the global search result cache. A value of `0` indicates that there is no expiration time. The default is `0` if the `LDAP_CLIENT_CACHE_TTL` environment variable is not defined.

LDAP_DEBUG
Specifies LDAP trace options. The value for `LDAP_DEBUG` is a mask that you can specify in the following ways:

- A decimal value (for example, `32`)
- A hexadecimal value (for example, `x20` or `X20`)
- A keyword (for example, `FILTER`)
- A construct of these values using plus and minus signs to indicate inclusion or exclusion of a value.

For more information about the LDAP trace options, see "Enabling tracing" on page 242.

LDAP_DEBUG_FILENAME
Specifies the fully qualified name of the LDAP trace output file. If this environment variable is not defined the trace output is written to `stdout`. The trace file is not used if LDAP tracing is not active.

The current process identifier is included as part of the trace file name when the name contains a percent sign (`%`).

Example: If `LDAP_DEBUG_FILENAME` is set to `/tmp/ldap.%.trc` and the current process identifier is 247, the trace file name is `/tmp/ldap.247.trc`.

Guideline: The trace file name should include a percent sign if the application creates extra processes that inherit environment variables because the trace output can be corrupted if multiple processes use the same trace file.

LDAP_ERROR_LOGGING
Specifies how error messages are logged. The following values can be specified:

STDOUT Error messages are written to standard output as specified by the `LDAP_STDOUT_FILENAME` environment variable.

STDERR Error messages are written to standard error as specified by the `LDAP_STDERR_FILENAME` environment variable.

BOTH Error messages are written to both standard output and to standard error.

If this environment variable is not defined error messages are written to standard error.

LDAP_EXC_ABEND_DUMP
LDAP provides its own version of TRY/CATCH for handling MVS™ abends. This support uses the LE condition handler support to intercept abends on a stack frame basis and continues execution within LDAP instead of ending the application. Because the abend is handled by LDAP, LE does not generate a dump for the error.

If you want a dump, set the `LDAP_EXC_ABEND_DUMP` environment variable to 1. This setting causes the LDAP condition handler to call the **cdump()** service to dump the current thread before resuming the failing routine. The **cdump()** service calls the LE **CEE3DMP** service to format the activation

stack and then calls the OS SNAP service to dump the virtual storage. The formatted activation stack is written to the data set or file identified by the CEEDUMP DD statement. If the CEEDUMP DD statement is not defined, the formatted activation stack is placed in the current directory unless LE has been instructed to use a different directory by the _CEE_DMPTARG environment variable. The virtual storage dump is written to the data set or file identified by the CEESNAP DD statement. No virtual storage dump is generated if the CEESNAP DD statement is not defined.

LDAP_SERVER_INFO_CONF
Specifies the name of the LDAP server information file that is used by the **ldap_init()** and **ldap_ssl_init()** routines when no host name is supplied as part of the LDAP URL. If this environment variable is not defined the file name defaults to /etc/ldap/ldap_server_info.conf.

LDAP_SERVER_INFO_CONF
Specifies the name of the LDAP server information file that is used by the **ldap_init()** and **ldap_ssl_init()** routines when no host name is supplied as part of the LDAP URL. If this environment variable is not defined the file name defaults to /etc/ldap/ldap_server_info.conf.

LDAP_SSL_CIPHER_FORMAT
Specifies the SSL cipher suites format. Specify CHAR2 to use the 2-byte cipher specifications that are defined by the z/OS System SSL environment variable GSK_V3_CIPHER_SPECS. This is the default. Specify CHAR4 to use the 4-byte cipher specifications that are defined by the z/OS System SSL environment variable GSK_V3_CIPHER_SPECS_EXPANDED.

LDAP_STDOUT_FILENAME
Specifies the fully qualified name of the file to receive standard output messages generated using LDAP message services. If this environment variable is not defined messages are written to stdout.

LDAP_VERSION
Specifies the LDAP protocol version that is used when the application does not set an explicit protocol version. Valid values are 2 and 3. The default is 3 for the **ldap_init()** and **ldap_ssl_init()** routines and 2 for the **ldap_open()** routine. The default is used if the environment variable is not defined or is not set to a valid value.

LDAP_V2_WIRE_FORMAT
Specifies the LDAP protocol version 2 attribute value format when an explicit wire format is not set by the application. Valid values are UTF-8 (or UTF8) and ISO8859-1. The default is UTF-8 if this environment variable is not defined or is not set to a valid value.

LOCALDOMAIN
Specifies the local DNS domain name. If this environment variable is not defined the DNS domain name is obtained from the DNS name resolver configuration file. The LOCALDOMAIN environment variable is used by the system name resolver routines and by LDAP.

RESOLVER_CONFIG
Specifies the fully qualified name of the DNS name resolver configuration file. If this environment variable is not defined the name resolver configuration file defaults to /etc/resolv.conf. The RESOLVER_CONFIG environment variable is used by the system name resolver routines and by LDAP.

For information about the contents of the name resolver configuration file, see "Name resolver configuration file" on page 244.

SOCKS_CONF
Specifies the fully qualified name of the SOCKS configuration file to be used by the LDAP client run time. A SOCKS server is not used if the SOCKS_CONF environment variable or the SOCKS_SERVER environment variable is defined. The SOCKS_CONF environment variable takes precedence if both SOCKS_CONF and SOCKS_SERVER are defined.

SOCKS_PASSWORD
Specifies the SOCKS password to be used when connecting to the LDAP server through a SOCKS server. A SOCKS user name and password are required when using the SOCKS version 5 protocol and the SOCKS server is configured to require user authentication. An unauthenticated SOCKS connection is used if the SOCKS user name and password are not set. Note authentication for the SOCKS connection is separate from the bind authentication for the LDAP server. The SOCKS user name and password are not used for the SOCKS version 4 protocol. The SOCKS_PASSWORD environment variable is not used if the SOCKS_USERNAME environment variable is not also defined.

SOCKS_SERVER
Specifies the SOCKS servers to be used by the LDAP client run time as a comma-separated list of servers. A SOCKS server is not to be used when the SOCKS_CONF environment variable or the SOCKS_SERVER environment variable is defined. The SOCKS_CONF environment variable takes precedence if both SOCKS_CONF and SOCKS_SERVER are defined. Each SOCKS server is specified as *host:port*. The host is a DNS name, an IPv4 address in dotted decimal format, or an IPv6 address in colon-separated format that is enclosed in square brackets. The port defaults to 1080 if it is not specified.

SOCKS_USERNAME
Specifies the SOCKS user name to be used when connecting to the LDAP server through a SOCKS server. A SOCKS user name and password are required when using the SOCKS version 5 protocol and the SOCKS server is configured to require user authentication. An unauthenticated SOCKS connection is used if the SOCKS user name and password are not set. Note authentication for the SOCKS connection is separate from the bind authentication for the LDAP server. The SOCKS user name and password are not used for the SOCKS version 4 protocol. The SOCKS_USERNAME environment variable is not used if the SOCKS_PASSWORD environment variable is not also defined.

SOCKS_VERSION
Specifies the SOCKS protocol version. Valid values are 4 and 5. The default is 4. However, the SOCKS version 5 protocol is always used when the LDAP server address is an IPv6 address, because the SOCKS version 4 protocol does not support IPv6 addresses. You can set the SOCKS_VERSION environment variable to 5 to cause the LDAP client run time to always use the SOCKS version 5 protocol.

Using SSL and TLS protected communications

The LDAP client can use Secure Socket Layer (SSL) or Transport Layer Security (TLS) to protect client communications using one of the following methods:

- To use SSL for only data integrity and confidentiality:
  - Initialize the SSL client runtime using **ldap_ssl_client_init()**.

  - Initialize the LDAP handle using **ldap_ssl_init()**.
  - Bind to the server using **ldap_simple_bind()** or **ldap_sasl_bind()**.

    Note: This method requires separate ports for SSL and non-SSL connections.
- To use SSL for both user authentication and for data integrity and confidentiality:
  - Initialize the SSL client runtime using **ldap_ssl_client_init()**.
  - Initialize the LDAP handle using **ldap_ssl_init()**.
  - Bind to the server using **ldap_sasl_bind()** with the `EXTERNAL` mechanism.

    Note: This method requires separate ports for SSL and non-SSL connections.
- To use SSL for data integrity and confidentiality for only a portion of the session:
  - Initialize the SSL client runtime using **ldap_ssl_client_init()**.
  - Initialize the LDAP handle using **ldap_init()**.
  - Bind to the server using **ldap_simple_bind()** or **ldap_sasl_bind()** with the `EXTERNAL` mechanism.
  - Initiate® TLS for the connection using **ldap_start_tls_s_np()**. (This step does not change the authentication method established for this connection. The established authentication method remains in place.)
  - Optionally, you can rebind with **ldap_sasl_bind()**, any time after issuing **ldap_start_tls_s_np()**, to switch to SSL authentication.
  - After the secure portion of the session completes, discontinue TLS using **ldap_stop_tls_s_np()**.

    Note: Using this method, the LDAP server can handle both SSL and non-SSL connections using a single port.

The **ldap_ssl_client_init()** and **ldap_ssl_init()** routines are used to start a secure connection to the LDAP server. Alternatively, the **ldap_start_tls_s_np()** routine can be used to start secure communications after a non-secure connection is established with the LDAP server.

To use SSL or TLS protected communications, the LDAP client needs access to a key database, SAF key ring, or PKCS #11 token. A key database is stored in a file that is accessible to the LDAP client and is created and maintained by the **gskkyman** command. A SAF key ring is stored in the external security manager and is created and maintained by the external security manager. (RACF provides the **RACDCERT** command.) A PKCS #11 token is stored and protected by ICSF. The **gskkyman** utility or the **RACDCERT** command provided by RACF can be used to create or modify PKCS #11 tokens. ICSF uses the CRYPTOZ SAF class to determine if the issuer of the **gskkyman** utility or the **RACDCERT** command is permitted to perform the operation against a z/OS PKCS #11 token. See *z/OS Cryptographic Services System SSL Programming* for more information about the **gskkyman** utility and *z/OS Security Server RACF Command Language Reference* for more information about the **RACDCERT** command. The key database, SAF key ring, or PKCS #11 token must contain the root certificate for the certification chain of the LDAP server's certificate. If the LDAP server is using a self-signed certificate, the client key database, SAF key ring, or PKCS #11 token must also contain this self-signed certificate. If the LDAP server is configured for client and server authentication and the LDAP client wants to use client authentication, the LDAP client must have its own certificate and this certificate and its certification chain must be stored in the key database, SAF key ring, or PKCS #11 token.

Using the socksified client

The LDAP client can be used to contact LDAP servers through a SOCKS server. The LDAP client has been *socksified* so that SOCKS Version 4 and SOCKS Version 5 servers can be used to connect to LDAP servers across firewalls on which a SOCKS server is running. To connect to an LDAP server through a SOCKS server, the LDAP client must be provided with the location of the SOCKS servers in your environment. This can be done in one of two ways:

- Through the SOCKS_SERVER environment variable
- Through the SOCKS_CONF environment variable along with the specified SOCKS configuration file.

Using the SOCKS_SERVER environment variable allows you to specify the locations of the SOCKS servers. All connections that are made by the LDAP client runtime then use the specified SOCKS servers. The SOCKS_SERVER environment variable is specified as a comma-separated list of SOCKS servers. Each SOCKS server is specified in the following format:

```
host:port
```

where:

host A DNS name, an IPv4 address in dotted decimal format, or an IPv6 address in colon-separated format enclosed in square brackets.

port This defaults to 1080 if it is not specified.

Examples:

```
export SOCKS_SERVER=9.14.33.90,9.130.25.36:8080
export SOCKS_SERVER=[FEC0::F4F7:0:0:7442:7501]:1080
export SOCKS_SERVER=mysockserver.mycompany.com:1075
```

Using the SOCKS_CONF environment variable allows you to specify the name of a SOCKS configuration file.

Example:

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

If the SOCKS_SERVER and SOCKS_CONF environment variables are not set, all connections are assumed to be direct. If both the SOCKS_SERVER and SOCKS_CONF environment variables are set, the SOCKS_CONF environment variable takes precedence.

Rules: The following are some rules for the SOCKS configuration file:

- The contents of the file must be in the IBM-1047 code page.
- The maximum line length is 1023 characters. Longer lines are truncated.
- Blank lines are ignored.
- Comment lines must have a # as the first non-blank character.
- The keywords and their values are not case-sensitive except for the values for the USERNAME and PASSWORD keywords. Whether the USERNAME and PASSWORD values are case-sensitive depends on the SOCKS server. The LDAP client run time sends the values as read from the SOCKS configuration file when authenticating with the SOCKS server.
- Entries that are not recognized or not valid are ignored.

You can use the following keywords in the SOCKS configuration file:

VERSION

The VERSION keyword sets the SOCKS protocol version. It must be 4 or 5. The version remains in effect until the next VERSION keyword. The initial value for the SOCKS protocol version is set by the SOCKS_VERSION environment variable.

```
VERSION number
```

USERNAME

The USERNAME keyword sets the SOCKS authentication user name. This user name is used for the SOCKS version 5 protocol to authenticate the connection with the SOCKS server. The user name remains in effect until the next USERNAME keyword. The initial value for the SOCKS user name is set by the SOCKS_USERNAME environment variable.

```
USERNAME name
```

PASSWORD

The PASSWORD keyword sets the SOCKS authentication password. This password is used for the SOCKS version 5 protocol to authenticate the connection with the SOCKS server. The password remains in effect until the next PASSWORD keyword. The initial value for the SOCKS password is set by the SOCKS_PASSWORD environment variable.

```
PASSWORD password
```

SOCKD

The SOCKD keyword tells the SOCKS client which SOCKS server or servers to use. The SOCKS protocol version is obtained from the most recent VERSION keyword. If there is no VERSION keyword preceding the SOCKD keyword, the SOCKS protocol version is 4 if the LDAP server address is an IPv4 address and 5 if the LDAP server address is an IPv6 address. An unauthenticated SOCKS connection is always used for the SOCKS version 4 protocol. An authenticated SOCKS connection is used for the SOCKS version 5 protocol if the USERNAME and PASSWORD keywords were specified before the SOCKD keyword. Otherwise, an unauthenticated SOCKS connection is used.

```
SOCKD @= server-list destination-address destination-mask
```

For compatibility with other implementations, the space can be omitted between the SOCKD keyword and the server list.

```
SOCKD@= server-list destination-address destination-mask
```

DENY The DENY keyword tells the SOCKS client which IP address or addresses it should refuse.

```
DENY destination-address destination-mask
```

DIRECT

The DIRECT keyword tells the SOCKS client that it should bypass the SOCKS server for the given IP address or addresses.

```
DIRECT destination-address destination-mask
```

where:

server-list

Consists of one or more comma-separated SOCKS server specifications. Specify each SOCKS server as *host:port*,

where:

host A DNS name, an IPv4 address in dotted decimal format, or an IPv6 address in colon-separated format enclosed in square brackets.

port This defaults to 1080 if it is not specified.

destination-address
An IPv4 address in dotted decimal format or an IPv6 address in colon-separated format.

destination-mask
An IP address in the same format as *destination-address* (IPv4 or IPv6). An IPv6 value is not enclosed in square brackets when used for *destination-address* or *destination-mask* because there is no ambiguity with a port specification.

Matching is performed by **AND**ing the LDAP server address with the destination mask and comparing the result to the destination address. The first matching line in the configuration file is used. Therefore, if you list the SOCKD keyword before the DIRECT or DENY keywords, all connections that match the SOCKD line go through the SOCKS server even though there is another matching line in the configuration file.

Example: The following is a sample SOCKS configuration file:

```
#####################################################################
# Sample SOCKS Configuration File
#
# Entirely blank lines are ignored.
# Lines with # in the first column are also ignored.
#
#  DENY dst_addr dst_mask
#  DIRECT dst_addr dst_mask
#  VERSION 5
#  USERNAME myname
#  PASSWORD mypw
#  SOCKD @=serverlist dst_addr dst_mask
#
# On connect, each line is processed in order and the first line
# that matches is used. If no line matches, the address is assumed
# to be direct.
#
# In order to cause all non-specified addresses to fail, place the
# following line at the end of the file:
#
#  DENY 0.0.0.0 0.0.0.0
#
# In this example, we are on network 192.168.100.x and the
# socks server is on the 192.168.100.205 system. All LDAP
# traffic to systems on the 192.168.100 net will be connected
# directly, while traffic to all other addresses will be
# through the SOCKS server.
#
#####################################################################

DIRECT 192.168.100.0 255.255.255.0
SOCKD @=192.168.100.205 0.0.0.0 0.0.0.0
```

Enabling tracing

Tracing can be enabled in the LDAP programming interface. Any change to trace options is global and affects all LDAP handles. There are two methods to enable tracing:

1. The first method is to use the **ldap_set_option()** API, specifying the option to be set as LDAP_OPT_DEBUG or LDAP_OPT_DEBUG_STRING. Once a new debug level is set using this method, the debug level that is specified with the LDAP_DEBUG environment variable is no longer in effect.

Example: To enable all trace classes using the **ldap_set_option()** API, specify one of the following:

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

or

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

The value that is specified for LDAP_OPT_DEBUG_STRING is a string that can have the same values as the LDAP_DEBUG environment variable. The call to **ldap_set_option()** can occur at any point after calling **ldap_init()** and before calling **ldap_unbind()** or **ldap_unbind_s()** to set or change debug options.

2. The second method for enabling tracing is to set the LDAP_DEBUG environment variable. The value for LDAP_DEBUG is a mask that you can specify as follows:
   - A decimal value (for example, 32).
   - A hexadecimal value (for example, x20, X20, 0x20, or 0X20)
   - A keyword (for example, FILTER)
   - A construct of those values using plus and minus signs to indicate inclusion or exclusion of a value. For example:
     - '32768+8' is the same as specifying 'x8000+x8', or 'ERROR+CONNS'
     - '2147483647-16' is the same as specifying 'x7FFFFFFF-x10' or 'ANY-BER'
     - By beginning the debug level with a minus sign, you can deactivate debug collection for a debug level. For example, "-CONNS" modifies an existing debug level by deactivating connection traces.
     - By beginning the debug level with a plus sign, you can activate debug collection for a debug level. For example, "+CONNS" modifies an existing debug level by activating connection traces.

   Note: Specifying the debug level using decimal or hex values with a plus or minus sign is not necessarily the same as specifying the sum or difference as the debug level. For example, specifying '7+1' activates the 'TRACE', 'PACKETS', and 'ARGS' debug levels, while specifying '8' activates only the 'CONNS' debug level. Similarly, specifying '16-1' activates only the 'BER' debug level, while specifying '15' activates 'TRACE', 'PACKETS', ARGS', and 'CONNS'.

   Restrictions: To enable or change tracing using this method, the LDAP_DEBUG environment variable must be set or changed before the client run time is first initialized. Later changes to the value of LDAP_DEBUG have no effect on the debug level of the client. If the debug level is set or changed using the LDAP_OPT_DEBUG or LDAP_OPT_DEBUG_STRING option, the debug level that is specified with the LDAP_DEBUG environment variable is no longer in effect.

   The LDAP trace routine uses the IBM-1047 code page when writing text data to the trace data set. The trace output is written to stdout unless the LDAP_DEBUG_FILENAME environment variable is defined. If the application creates additional processes, specify % as part of the trace file name. This causes the % to be replaced by the current process identifier, therefore, generating a unique file name for each process. Failure to do this can cause the trace file to be corrupted because locking is done on a process basis.

   Example: The following example shows the use of % in the trace file name.

   ```
   export LDAP_DEBUG_FILENAME=/tmp/myapp.%.trc
   ```

   Table 5 on page 244 lists the debug levels and related decimal, hexadecimal, and keyword values. Keywords can be abbreviated using the uppercase characters for each keyword.

Table 5. LDAP debug levels

Keyword	Decimal	Hexadecimal	Description
OFF	0	0x00000000	No debugging
TRACe	1	0x00000001	Entry and exit from routines
PACKets	2	0x00000002	Packet activity
ARGS	4	0x00000004	Data arguments from requests
CONNs	8	0x00000008	Connection activity
BER	16	0x00000010	Encoding and decoding of data, including ASCII and EBCDIC translations, if applicable
FILTer	32	0x00000020	Search filters
MESSage	64	0x00000040	Messaging subsystem activities and events
ACL	128	0x00000080	Access Control List activities
STATs	256	0x00000100	Operational statistics
THREad	512	0x00000200	Threading activities
REPLication	1024	0x00000400	Replication activities
PARSe	2048	0x00000800	Parsing activities
PERFormance	4096	0x00001000	Performance statistics
SDBM	8192	0x00002000	Backend activities (SDBM)
REFErral	16384	0x00004000	Referral activities
ERROr	32768	0x00008000	Error conditions
SYSPlex	65536	0x00010000	Sysplex/WLM activities
MULTIServer	131072	0x00020000	Multi-server activities
LDAPBE	262144	0x00040000	Connection between a frontend and a backend
STRBuf	524288	0x00080000	UTF-8 support activities
TDBM	1048576	0x00100000	Backend activities (TDBM)
SCHEma	2097152	0x00200000	Schema support activities
BECApabilities	4194304	0x00400000	Backend capabilities
CACHe	8388608	0x00800000	Cache activities
INFO	16777216	0x01000000	General processing information
LDBM	33554432	0x02000000	Backend activities (LDBM)
PLUGin	67108864	0x04000000	Plug-in extension activities
ANY	2147483647	0x7FFFFFFF	All levels of debug
ALL	2147483647	0x7FFFFFFF	All levels of debug
Note: The minimum abbreviation for each keyword is shown in uppercase letters.			

Name resolver configuration file

The name resolver configuration file is used by the LDAP client when it must locate an LDAP server. The resolver configuration file name is specified by the `RESOLVER_CONFIG` environment variable and defaults to `/etc/resolv.conf`.

Rules: The resolver configuration file must follow these rules:

- Each line in the configuration file has a maximum length of 255 characters and consists of a keyword and a value that is separated by one or more white space characters.
- Comment lines begin with # or **;**.
- Blank lines are ignored.
- The configuration file must be in the IBM-1047 code page.
- The keywords and their values are not case-sensitive.
- The NSINTERADDR and NAMESERVER keywords can be specified on multiple lines and the name server list includes all the specified addresses.
- The SEARCH keyword can be specified on multiple lines and the domain list includes all the specified names.
- Keywords other than NSINTERADDR, NAMESERVER, and SEARCH should be specified once. If the resolver configuration file specifies one of these other keywords more than once, the LDAP name resolver uses the last occurrence.

The LDAP name resolver uses the following keywords in the resolver configuration file and ignores any other values:

DOMAIN Specifies the default DNS domain name. The DOMAIN and DOMAINORIGIN keywords are the same and can be used interchangeably. The DOMAIN, DOMAINORIGIN, and SEARCH keywords are mutually exclusive. The search list specified by the SEARCH keyword is deleted if the DOMAIN keyword follows the SEARCH keyword. This keyword is ignored if the `LOCALDOMAIN` environment variable is defined.

DOMAINORIGIN
Specifies the default DNS domain name. The DOMAIN and DOMAINORIGIN keywords are the same and can be used interchangeably. The DOMAIN, DOMAINORIGIN, and SEARCH keywords are mutually exclusive. The search list specified by the SEARCH keyword is deleted if the DOMAINORIGIN keyword follows the SEARCH keyword. This keyword is ignored if the `LOCALDOMAIN` environment variable is defined.

NAMESERVER
Specifies the network address of a DNS name server. An IPv4 address is specified in dotted decimal format. An IPv6 address is specified in colon-hexadecimal format. The NSINTERADDR and NAMESERVER keywords are the same and can be used interchangeably.

NSINTERADDR
Specifies the network address of a DNS name server. An IPv4 address is specified in dotted decimal format. An IPv6 address is specified in colon-hexadecimal format. The NSINTERADDR and NAMESERVER keywords are the same and can be used interchangeably.

NSPORTADDR
Specifies the well-known port for the DNS name servers. This is a decimal number and defaults to `53`.

RESOLVERTIMEOUT
Specifies the number of seconds to wait for an answer. This is a decimal number and defaults to `5` seconds.

RESOLVERUDPRETRIES
Specifies the number of retries when using UDP (User Datagram Protocol). This is a decimal number and defaults to `1`.

SEARCH Specifies one or more DNS domain names. (Multiple domain names are

separated by white space characters.) These domains are searched in order when looking for an LDAP resource name. The DOMAIN, DOMAINORIGIN, and SEARCH keywords are mutually exclusive. The default domain is set to the first domain specified by the SEARCH keyword and replaces a value specified by the DOMAIN or DOMAINORIGIN keyword. This keyword is ignored if the `LOCALDOMAIN` environment variable is defined.

Example: The following is a sample name resolver configuration file:

```
######################################################################
#  Sample name resolver configuration file                           #
######################################################################
TCPIPJobname CS390IP
DatasetPrefix SHR.CS390IP
ResolveVIA UDP
ResolverTimeout 5
NameServer 9.130.77.115
NameServer 9.130.40.252
NameServer 9.130.40.242
Domain endicott.ibm.com
```

For more information about the contents of the name resolver configuration file, see *z/OS V2R2.0 Communications Server: IP Configuration Reference*.

LDAP server information file

Information about LDAP server locations and capabilities can be saved in a server information file. The **ldap_server_locate()** routine can read this server information file when the LDAP client must locate a server. You can create this file using the **ldap_server_conf_save()** routine, or you can create and maintain it manually.

Guideline: Use the **ldap_server_conf_save()** routine to create the server information file.

Rules: If you choose to create and manually maintain the server information file, follow these rules:

- The contents of the file must be in the IBM-1047 code page.
- The maximum line length is 1023 characters.
- Blank lines are ignored.
- Comment lines must have a # as the first non-blank character.
- All parameters are positional.
- The first non-comment line must contain the expiration time for the file. This time is a decimal number and is expressed as a POSIX time value (number of seconds since January 1, 1970 UTC). A value of `0` indicates that the file does not expire.
- Each line following the server-information-file expiration time represents an LDAP server definition.
- Incorrect numeric values are treated as zero values.

Each LDAP server is defined by a line in the following format:

service domain host [priority [weight [port [replica [security [naming [vendor [general]]]]]]]]

The fields are positional and are defined as follows:

service
Specifies the service name and is formed by combining the service key and the optional eNetwork domain name as *service_key*.*enetwork_domain*. This field must be specified.

domain
Specifies the DNS domain name for the LDAP server. This field must be specified.

host
Specifies the fully qualified DNS name of the LDAP server host. This field must be specified.

priority
A decimal number that specifies the priority that is assigned to the LDAP server. The **ldap_server_locate()** routine returns the server list that is ordered by priority. (The priority decreases as the priority number increases.) Specify `0` for the priority if the servers are not to be ordered by priority. This field can be omitted if all the following fields are also omitted, in which case the priority defaults to `0`.

weight
A decimal number that specifies the weight that is assigned to the LDAP server within the priority classification. The weight is used as a load-balancing mechanism and indicates the capacity of the LDAP server relative to other servers with the same priority value. Servers with a larger weight are selected more often than servers with a smaller weight. Specify `0` for the weight if load balancing is not needed. This field can be omitted if all the following fields are also omitted, in which case the weight defaults to `0`.

port
A decimal number that specifies the port to use to contact the LDAP server. This field can be omitted if all the following fields are also omitted, in which case the port defaults to `389`.

replica
Specifies whether the LDAP server is a master or a replica. Specify `1` to indicate master and `2` to indicate replica. This field can be omitted if all the following fields are also omitted, in which case *replica* defaults to `0` (replica type not specified).

security
Specifies the connection security mechanism. Specify `1` to indicate non-SSL and `2` to indicate SSL. This field can be omitted if all the following fields are also omitted, in which case the security defaults to `0` (security type not specified).

naming
Specifies the naming context that is supported by the server. The string must be enclosed in double quotation marks if it contains any white space characters. A double quotation mark or backslash in the string must be escaped using a backslash. Multiple server entries must be defined if the LDAP server supports more than one naming context. This field can be omitted if all the following fields are also omitted. Otherwise, it must be specified as `""` if there is no naming context for the LDAP server.

vendor
Specifies vendor information for the LDAP server. The string must be enclosed in double quotation marks if it contains any white space characters. A double quotation mark or backslash in the string must be escaped using a backslash.

This field can be omitted if the following field is also omitted. Otherwise, it must be specified as "" if there is no vendor information for the LDAP server.

general
Specifies general information for the LDAP server. The string must be enclosed in double quotation marks if it contains any white space characters. A double quotation mark or backslash in the string must be escaped using a backslash. This field can be omitted or specified as "" if there is no general information for the LDAP server.

Example of a server information file

Following is a sample server information file:

```
####################################################################
#  Sample LDAP local configuration file                            #
####################################################################
0
ldap.research endicott.ibm.com sysa.endicott.ibm.com 0 0 389 1 1
ldap.research endicott.ibm.com sysa.endicott.ibm.com 0 0 636 1 2
ldap.research endicott.ibm.com backup.endicott.ibm.com 5 0 389 1 1
ldap.research endicott.ibm.com backup.endicott.ibm.com 5 0 636 1 2
_ldap endicott.ibm.com sysb.endicott.ibm.com 0 0 636 1 2 "o=ibm,c=us"
_ldap endicott.ibm.com sysb.endicott.ibm.com 0 0 636 1 2 "dc=ibm,dc=com"
_ldap.endicott.ibm.com replica.endicott.ibm.com 0 0 636 2 2 "o=ibm,c=us"
_ldap encoditt.ibm.com replica.endicott.ibm.com 0 0 636 2 2 "dc=ibm,dc=com"
```

The sysa.endicott.ibm.com and backup.endicott.ibm.com systems have LDAP servers that are part of the research eNetwork domain. The LDAP server on backup.endicott.ibm.com is used only if the LDAP server on sysa.endicott.ibm.com is not available. Note that there are two entries for each host: one for the non-SSL connection and the other for the SSL connection.

The sysb.endicott.ibm.com and replica.endicott.ibm.com systems have LDAP servers that are not part of an eNetwork domain. They support naming contexts "o=ibm,c=us" and "dc=ibm,dc=com". The LDAP server on sysb.endicott.ibm.com is the master server and the LDAP server on replica.endicott.ibm.com is a replica server. Note that there are two entries for each host: one for naming context "o=ibm,c=us" and the other for naming context "dc=ibm,dc=com".

Publishing LDAP server information in DNS

If DNS is to be used to publish LDAP server information, the relevant DNS name server or servers must be configured with the appropriate SRV and TXT records that reflect the LDAP servers available in the enterprise. SRV records are used to identify the LDAP servers in the enterprise along with appropriate priority and weight values. TXT records are associated with each LDAP server host to specify the LDAP URL used to access the LDAP server on that host and to provide information about the capabilities of the LDAP server. If SRV records are not supported by the DNS name server, TXT records can be used to emulate the SRV records or a CNAME record can be used to point directly to a single LDAP server host.

Domain name service resource names have a maximum length of 255 characters and use the ISO8859-1 code page. LDAP converts character parameters that are supplied by the application from the local EBCDIC code page to the ISO8859-1 code page when sending a request to the domain name server, and then converts the name server response from the ISO8859-1 code page back to the local EBCDIC code page when returning the results to the application.

The domain name server list must either contain the name server that is authoritative for the zone containing the LDAP server information, or one of the domain name servers in the list must support recursion and forward the query to the authoritative name server.

The DNS lookup routine ignores unrecognized TXT records and TXT records containing syntax errors.

Using SRV and TXT records

The DNS lookup routine looks for SRV records first. If one or more servers are found, this server information is used and the second algorithm, which is based on TXT records that emulate SRV records, is not used. The use of SRV records for finding the address of servers is described in RFC 2052: *A DNS RR for specifying the location of services (DNS SRV)*. Proper use of SRV records permits the administrator to distribute a service across multiple hosts within a domain, to move the service from host to host without disruption, and to designate certain hosts as primary and others as alternates.

TXT records are simply character strings that are associated with a DNS resource name. LDAP uses TXT records to associate LDAP server information with a DNS host name. To implement the technique that is described in RFC 2052, the DNS name server must support both SRV and TXT records.

An SRV resource record (RR) has the following components:

service.protocol.domain ttl class **SRV** *priority weight port target*

The fields are positional and are defined as follows:

service
Symbolic name of the service. The service name is formed by concatenating the service key and the eNetwork domain name (if any). The LDAP client accepts either `ldap` or `_ldap` for the service key. The latest version of RFC 2052 recommends the use of `_ldap` instead of `ldap`.

protocol
Protocol used to access the service. The LDAP client accepts either `tcp` or `_tcp`. The latest version of RFC 2052 recommends the use of `_tcp` instead of `tcp`.

domain
Domain name associated with the resource record.

ttl
Time-to-live in seconds.

class
Class (must be `IN` for internet).

SRV
Indicates this is an SRV record.

priority
Service priority. LDAP servers are ordered by priority with the lower priority numbers ordered before the higher priority numbers. Set the priority to `0` if priority ordering is not wanted.

weight
Load balancing within the same priority. A higher weight number indicates that the server can handle more requests than a lower weight number. The probability that a server is ordered early in the list increases as the weight

increases. Set the weight to 0 if load balancing is not wanted. Otherwise, use nonzero values for all the weights within the same priority. (An SRV record with a weight of 0 has a low probability of being ordered before an SRV record with a nonzero weight).

port
The port assigned to the LDAP server. This value is ignored if the target address record has a service TXT record. If the port number is 0, the port is set to 389.

target
The name of the target address resource record (A, AAAA, or A6). The host name used to connect to the LDAP server is obtained from the service TXT record associated with this resource name. If there is no service TXT record defined for the target, the IP address is obtained from the address record.

A TXT record has the following format:

```
name TXT "string"
```

The fields are positional and are defined as follows:

name
Resource name associated with the TXT record.

TXT
Indicates this is a TXT record.

string
Text value.

A TXT record defining a non-SSL server connection has the following format:

```
name TXT "service:ldap://host-name[:port][/naming-context]"
```

The host name must be specified. The port defaults to 389 if it is not specified. A naming context can be specified to allow server entries to be selectively filtered based upon a distinguished name. Multiple service TXT records must be defined if more than one naming context is defined for a single LDAP server or if the LDAP server supports both SSL and non-SSL connections.

A TXT record defining an SSL server connection has the following format:

```
name TXT "service:ldaps://host-name[:port][/naming-context]"
```

The host name must be specified. The port defaults to 636 if it is not specified. A naming context can be specified to allow server entries to be selectively filtered based upon a distinguished name. Multiple service TXT records must be defined if more than one naming context is defined for a single LDAP server or if the LDAP server supports both SSL and non-SSL connections.

A TXT record defining a master LDAP server has the following format:

```
name TXT "ldaptype:master"
```

The last ldaptype TXT record encountered is used if more than one ldaptype TXT record is defined for the same target.

A TXT record defining a replica LDAP server has the following format:

```
name TXT "ldaptype:replica"
```

The last `ldaptype` TXT record encountered is used if more than one `ldaptype` TXT record is defined for the same target.

A TXT record defining server vendor information has the following format:

name **TXT "ldapvendor:***vendor-information***"**

The LDAP client does not use the vendor information but makes it available to the application. The last `ldapvendor` TXT record encountered is used if more than one `ldapvendor` TXT record is defined for the same target.

A TXT record defining general server information has the following format:

name **TXT "ldapinfo:***general-information***"**

The LDAP client does not use the general information but makes it available to the application. The last `ldapinfo` TXT record encountered is used if more than one `ldapinfo` TXT record is defined for the same target.

Example of DNS resource records

The following are the DNS resource records that correspond to the sample server information file described in "Example of a server information file" on page 248. These examples assume that the DNS name server database provides appropriate default values for the *ttl* and *class* fields, the resource record name can be omitted if it is the same as the preceding record, and the domain origin is `endicott.ibm.com`.

```
ldap.research.tcp SRV 0 0 0 sysa
                  SRV 5 0 0 backup
_ldap._tcp        SRV 0 0 0 sysb
                  SRV 0 0 0 replica
sysa              A   9.130.25.34
                  TXT "service:ldap://sysa.endicott.ibm.com:389"
                  TXT "service:ldaps://sysa.endicott.ibm.com:636"
backup            A   9.130.25.35
                  TXT "service:ldap://backup.endicott.ibm.com:389"
                  TXT "service:ldaps://backup.endicott.ibm.com:636"
sysb              A   9.130.36.4
                  TXT "service:ldaps://sysb.endicott.ibm.com:636/dc=ibm,dc=com"
                  TXT "service:ldaps://sysb.endicott.ibm.com:636/o=ibm,c=us"
                  TXT "ldaptype:master"
replica           A   9.130.36.5
                  TXT "service:ldaps://replica.endicott.ibm.com:636/dc=ibm,dc=com"
                  TXT "service:ldaps://replica.endicott.ibm.com:636/o=ibm,c=us"
                  TXT "ldaptype:replica"
```

Note that there are two `service` TXT records for `sysa.endicott.ibm.com` and `backup.endicott.ibm.com`, one for the non-SSL port and one for the SSL port. Similarly, there are two `service` TXT records for `sysb.endicott.ibm.com` and `replica.endicott.ibm.com`: one for naming context `"dc=ibm,dc=com"` and one for naming context `"o=ibm,c=us"`.

These LDAP servers could also be defined using a single `service` TXT record for each resource name. In this case, multiple SRV and host address records are needed. While it is preferable to use a single SRV record for each LDAP server, some implementations of the LDAP DNS support might require multiple SRV records with a single `service` TXT record for each resource name. The definitions would then be as follows:

```
ldap.research.tcp SRV 0 0 0 sysa
                  SRV 0 0 0 sysasec
                  SRV 5 0 0 backup
                  SRV 5 0 0 backupsec
_ldap._tcp        SRV 0 0 0 sysb1
                  SRV 0 0 0 sysb2
                  SRV 0 0 0 replica1
                  SRV 0 0 0 replica2
```

```
sysa            A   9.130.25.34
                TXT "service:ldap://sysa.endicott.ibm.com:389"
sysasec         A   9.130.25.34
                TXT "service:ldaps://sysa.endicott.ibm.com:636"
backup          A   9.130.25.35
                TXT "service:ldap://backup.endicott.ibm.com:389"
backupsec       A   9.130.25.35
                TXT "service:ldaps://backup.endicott.ibm.com:636"
sysb1           A   9.130.36.4
                TXT "service:ldaps://sysb.endicott.ibm.com:636/dc=ibm,dc=com"
                TXT "ldaptype:master"
sysb2           A   9.130.36.4
                TXT "service:ldaps://sysb.endicott.ibm.com:636/o=ibm,c=us"
                TXT "ldaptype:master"
replica1        A   9.130.36.5
                TXT "service:ldaps://replica.endicott.ibm.com:636/dc=ibm,dc=com"
                TXT "ldaptype:replica"
replica2        A   9.130.36.5
                TXT "service:ldaps://replica.endicott.ibm.com:636/o=ibm,c=us"
                TXT "ldaptype:replica"
```

Using TXT records to emulate SRV records

If no servers are found using SRV records, the search is repeated using TXT records to emulate SRV records. The previous example would be defined as follows using pseudo-SRV records:

Example:

```
ldap.research.tcp TXT "0 0 0 sysa.endicott.ibm.com."
                  TXT "5 0 0 backup.endicott.ibm.com."
_ldap._tcp        TXT "0 0 0 sysb.endicott.ibm.com."
                  TXT "0 0 0 replica.endicott.ibm.com."
sysa              A   9.130.25.34
                  TXT "service:ldap://sysa.endicott.ibm.com:389"
                  TXT "service:ldaps://sysa.endicott.ibm.com:636"
backup            A   9.130.25.35
                  TXT "service:ldap://backup.endicott.ibm.com:389"
                  TXT "service:ldaps://backup.endicott.ibm.com:636"
sysb              A   9.130.36.4
                  TXT "service:ldaps://sysb.endicott.ibm.com:636/dc=ibm,dc=com"
                  TXT "service:ldaps://sysb.endicott.ibm.com:636/o=ibm,c=us"
                  TXT "ldaptype:master"
replica           A   9.130.36.5
                  TXT "service:ldaps://replica.endicott.ibm.com:636/dc=ibm,dc=com"
                  TXT "service:ldaps://replica.endicott.ibm.com:636/o=ibm,c=us"
                  TXT "ldaptype:replica"
```

Fully qualified host names (including the final period) should be used as the target on the pseudo-SRV records because, unlike SRV records, the DNS name server does not resolve them when providing the answer to the LDAP client. The LDAP client assumes that a relative name used as a target host name in a pseudo-SRV record is in the same domain as the resource name used to access the record.

Using CNAME records

If no servers are found using SRV records or pseudo-SRV records, the search is repeated using a single host entry designated by a CNAME record. This method allows a single LDAP server to be associated with a service name. The previous example could be represented as follows with a single LDAP server for each service name:

Example:

```
ldap.research.tcp CNAME sysa
_ldap._tcp        CNAME sysb
sysa              A   9.130.25.34
                  TXT "service:ldap://sysa.endicott.ibm.com:389"
                  TXT "service:ldaps://sysa.endicott.ibm.com:636"
```

```
sysb            A   9.130.36.4
                TXT "service:ldaps://sysb.endicott.ibm.com:636/dc=ibm,dc=com"
                TXT "service:ldaps://sysb.endicott.ibm.com:636/o=ibm,c=us"
                TXT "ldaptype:master"
```

ldap_server_locate() usage by ldap_init() and ldap_ssl_init()

The **ldap_init()** and **ldap_ssl_init()** routines are used to establish connections to LDAP servers. These routines accept a URL to identify the host and port of an LDAP server. The LDAP URL for a non-SSL connection is:

```
ldap://host:port/dn?attributes?scope?filter
```

and the LDAP URL for an SSL connection is:

```
ldaps://host:port/dn?attributes?scope?filter
```

where:

host Specifies the DNS host name of the LDAP server.

port Specifies the port number for the LDAP server and defaults to `389` for a non-SSL connection and `636` for an SSL connection.

dn Specifies a distinguished name used to select available LDAP servers that are based upon the defined naming contexts.

The *attributes*, *scope*, and *filter* values are ignored when binding to the LDAP server.

The **ldap_server_locate()** routine is called to locate the LDAP server if no host name is specified as part of the LDAP URL. The **ldap_server_locate()** routine searches the server information file followed by the DNS name server. The server information file is defined by the `LDAP_SERVER_INFO_CONF` environment variable and defaults to `/etc/ldap/ldap_server_info.conf` if this environment variable is not defined.

The following URL causes the **ldap_init()** routine to call the **ldap_server_locate()** routine to locate an LDAP server that supports naming context `"o=IBM,c=US"` using a non-secure (non-SSL) connection:

Example:

```
ldap:///cn=Scott,o=IBM,c=US
```

Chapter 5. LDAP client utilities

Several utility programs are provided that implement some of the LDAP APIs. These utilities provide a way to add, compare, modify, search, and delete entries in any server accepting LDAP protocol requests.

Each of the following utilities can be run from the z/OS shell or TSO:

- **ldapchangepwd**
- **ldapcompare**
- **ldapdelete**
- **ldapadd**
- **ldapmodify**
- **ldapmodrdn**
- **ldapsearch**

See "Using the LDAP client utilities" on page 256 for information about how the utilities authenticate to the targeted LDAP server.

Restriction: This topic does not contain programming interface information.

Running the LDAP client utilities in the z/OS shell

To run any of these utilities in the shell, some environment variables must be set properly. Ensure that `/bin` is included in the `PATH` environment variable. Set `STEPLIB` to `SYS1.SIEALNKE` if that data set is not in the LNKLIST.

Running the LDAP client utilities in TSO

The LDAP client utilities can be run from TSO. To do this, some elements of the environment must be set up to locate the LDAP programs. Following are the steps to do this:

1. The PDS (SYS1.SIEALNKE) where the LDAP utility load modules were installed must be accessible through **LNKLIB, LPALIB**, or specified on the **TSOLIB** command.

   ```
   tsolib act dsn('SYS1.SIEALNKE')
   ```

2. The PDS (*GLDHLQ*.SGLDEXEC) containing the CLISTs needed to run the utilities must be available in SYSEXEC:

   ```
   alloc f(SYSEXEC) da('GLDHLQ.SGLDEXEC')
   ```

3. The default environment variables file for the utilities can be changed by creating a data set to hold the environment variables and then by using the TSO **alloc** command as shown:

   ```
   alloc f(ENVVAR) da('datasetname')
   ```

If you are using the utilities in interactive mode (for example, reading Dns, `changetype:` lines, and so on, from standard input), you can break out of interactive mode by pressing the PA1 key. Doing this returns the TSO session to the READY prompt. This is like pressing Ctrl+C keys in z/OS UNIX System Services.

After this setup is complete, running these utilities follows the same syntax as would be used if running them in z/OS, except that the program names are eight characters or less. To run these utilities from TSO, use the following names:

Table 6. Names for running LDAP client utilities from TSO

z/OS shell name	TSO name
ldapadd	ldapadd
ldapchangepwd	ldapchpw
ldapcompare	ldapcmpr
ldapdelete	ldapdlet
ldapmodify	ldapmdfy
ldapmodrdn	ldapmrdn
ldapsearch	ldapsrch

Using the LDAP client utilities

The **ldapadd, ldapchangepwd, ldapcompare, ldapdelete, ldapmodify, ldapmodrdn**, and **ldapsearch** utilities support authenticating with LDAP version 2 or 3 to the targeted LDAP server. By default, the client utilities use LDAP version 3 unless **-V 2** is specified on the command line.

If LDAP version 2 is used, the client utilities invoke the **ldap_sasl_bind_s()** routine to perform a simple or anonymous bind (authentication) to the targeted LDAP server.

If LDAP version 3 is used, the client utilities invoke the **ldap_sasl_bind()** routine to perform a simple, CRAM-MD5, DIGEST-MD5, GSSAPI (Kerberos), or EXTERNAL bind and send the password policy control (`1.3.6.1.4.1.42.2.27.8.5.1`) as a non-critical control to the targeted LDAP server. (If an anonymous bind is done while in LDAP version 3, the client utilities do not invoke a bind routine). The bind mechanism used by the client utilities is determined by the **-m** or **-S** parameter.

When the bind routine is invoked, several results can be returned. Following are bind results by using various combinations of user IDs and passwords:

1. If a null or zero length DN is specified, the user receives unauthenticated access.
2. If a non-null, nonzero length DN is specified, a password must also be specified.
   - If the DN falls outside the scope of the suffixes that are managed by the server, the DN must match one of the **adminDN, masterServerDN**, or **peerServerDN** configuration file options specified in the server configuration file, and the password must match the corresponding **adminPW, masterServerPW**, or **peerServerPW** configuration file option. In this case, the user is bound as the LDAP server root administrator or as the master or peer replica administrator.
   - If the DN falls within the scope of a suffix managed by the server, then there must be an entry in the server directory for that DN. The password specified by the user must match a password associated with the entry. The user is then bound with that identity. If the DN also matches one of the **adminDN, masterServerDN**, or **peerServerDN** configuration file options specified in the server configuration file, then the user is bound as the LDAP server root

administrator or as the master or peer replica administrator. If the DN has been assigned one or more administrative roles, then the user is bound with those administrative roles. See Administration groups and roles in *z/OS IBM Tivoli Directory Server Administration and Use for z/OS* for more information about administrative roles.

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

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

If the targeted LDAP server supports the password policy control and the user specified during the simple, CRAM-MD5, or DIGEST-MD5 bind is subject to password policy on the LDAP server, the LDAP server returns a password policy control response to the client utilities. If a password policy control response is returned by the targeted LDAP server, the client utilities parse and display the password policy warning or error message. The following are examples of password policy warnings and errors displayed by the client utilities after retrieving the bind result message.

1. This example shows the results of a simple, CRAM-MD5, or DIGEST-MD5 authentication when the user's password is expired. In this example, the user does not successfully authenticate to the targeted LDAP server because the password has expired. The utility ends because authentication is not successful.

   ```
   ldap_sasl_bind: Credentials are not valid
   ldap_sasl_bind: additional info: R004196 The 'userpassword' attribute value has
    passed its maximum age of 999999 seconds (srv_pwd_bind_check:3412)
   ldap_sasl_bind: Error, password has expired
   ```

2. This example shows the results of a simple, CRAM-MD5, or DIGEST-MD5 authentication when the user's password must be changed after a reset and there is one grace login remaining. In this example, the user is authenticated to the targeted LDAP server because there are grace logins remaining. The utility continues running and attempts the operations specified in the input file, standard input, or on the command line.

   ```
   ldap_sasl_bind: Password must be changed
   ldap_sasl_bind: Warning, 1 grace logins remain
   continuing processing...
   ```

3. This example shows the results of a simple, CRAM-MD5, or DIGEST-MD5 authentication when the user's password is to expire in just over 10 days. In this example, the user is authenticated to the targeted LDAP server because the password has not yet expired. The utility continues running and attempts the operations specified in the input file, standard input, or on the command line.

   ```
   ldap_sasl_bind: Warning, time before password expiration is 900643
   ldap_sasl_bind: additional info: Time before password expiration is 10
    days and 10:10:43
   continuing processing...
   ```

   The first message indicates the password expiration in number of seconds while the second message converts the number of seconds into a more readable format.

If LDAP version 3 is used in the **ldapadd, ldapchangepwd, ldapcompare**, and **ldapmodify** utilities, the password policy control is also automatically sent as a non-critical control to the targeted LDAP server on each add, compare, and modify operation. If the user being added, compared, or modified is subject to password policy on that server, the LDAP server returns a password policy control response to the client utilities. The client utilities parse and display the password policy

warning or error message. The following are examples of password policy warnings and errors displayed by the client utilities after issuing the add, compare, or modify operation.

1. This example shows the results of modifying a user's password value after already being successfully authenticated to the targeted LDAP server. The modify operation failed because the new password value is not 8 characters or longer.

```
ldap_modify: Constraint violation
ldap_modify: additional info: R004194 The 'userpassword' attribute
 value requires a minimum of 8 characters (pwd_validate_password_quality:2542)
ldap_modify: Error, password is too short
```

2. This example shows the results of comparing a user's password value after authenticating to the targeted LDAP server. The compare operation failed because the user being compared has had their account locked.

```
ldap_compare: Credentials are not valid
ldap_compare: Error, account is locked
```

3. This example shows the results of adding an entry after already being successfully authenticated to the targeted LDAP server. The add operation failed because the password value specified did not abide by the password syntax checking on the targeted LDAP server.

```
ldap_add: Constraint violation
ldap_add: additional info: R004190 The 'userpassword' attribute value
 allows a maximum of 3 repeated characters (pwd_validate_password_quality:2508)
ldap_add: Error, password syntax is not valid
```

Note: When the client utilities display the password policy control response warning or error message, the LDAP routine called by the client utilities is indicated in the prefix of the output messages. The prefix of the output messages is everything before the colon (:). For example:

```
routine: message
```

Specifying a value for a file name

When running the **ldapadd, ldapchangepwd, ldapcompare, ldapdelete, ldapmodify, ldapmodrdn**, and **ldapsearch** utilities, the file option (**-f**) value can be specified as follows:

/pathname/filename
: Specifies the full path name of a file in the z/OS UNIX System Services file systems.

filename
: Specifies a path name that is relative to the current working directory of the LDAP client utility.

  Note: When running from batch, there is no current working directory that is defined. This format is not suggested.

"//'dataset.name'"
: Specifies the fully qualified name of a file that is stored in a sequential data set.

"//'dataset.name(member)'"
: Specifies the fully qualified name of a file that is stored in a partitioned data set.

SSL/TLS information for LDAP client utilities

The contents of a client's key database file is managed with the **gskkyman** utility. See *z/OS Cryptographic Services System SSL Programming* for information about the **gskkyman** utility. The **gskkyman** utility is used to define the set of trusted certificate authorities (CAs) that are to be trusted by the client. By obtaining certificates from trusted CAs, storing them in the key database file, and marking them as trusted, you can establish a trust relationship with LDAP servers that use certificates issued by one of the CAs that are marked as trusted.

If the LDAP servers accessed by the client use server authentication, it is sufficient to define one or more trusted root certificates in the key database file. With server authentication, the client can be assured that the target LDAP server has been issued a certificate by one of the trusted CAs. In addition, all LDAP transactions that flow over the SSL/TLS connection with the server are encrypted, including the LDAP credentials that are supplied on the **ldap_sasl_bind_s()** API.

For example, if the LDAP server is using a high-assurance VeriSign certificate, obtain a CA certificate from VeriSign, receive it into your key database file, and mark it as trusted. If the LDAP server is using a self-signed **gskkyman** server certificate, the administrator of the LDAP server can supply you with a copy of the server's certificate request file. Receive the certificate request file into your key database file and mark it as trusted.

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

SSL/TLS encrypts the key database file therefore either the key database password or a stash file must be specified on the **-P** parameter. If a stash file is specified, it must be specified in the form `file://` followed immediately (no blanks in between) by the file specification of the stash file. See *z/OS Cryptographic Services System SSL Programming* for information about using the **gskkyman** utility to create a stash file.

Using RACF key rings

Alternately, LDAP supports the use of a RACF key ring. See Certificate/Key management in *z/OS Cryptographic Services System SSL Programming* for instructions on how to migrate a key database to RACF and how to use the **RACDCERT** command to protect the certificate and key ring.

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

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

Remember to refresh the RACF FACILITY class after doing the authorization:

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

After the RACF key ring is set up and authorized, specify the RACF key ring name for the **-K** *keyFile* option and do not specify the **-P** *keyFilePW* option.

Using PKCS #11 tokens

The LDAP client supports the use of PKCS #11 tokens. PKCS #11 tokens are stored and protected by ICSF. The **gskkyman** utility or the **RACDCERT** command can be used to create or modify PKCS #11 tokens. ICSF uses the CRYPTOZ SAF class to determine if the issuer of the **gskkyman** utility or the **RACDCERT** command is permitted to perform the operation against a z/OS PKCS #11 token. For information about using the **gskkyman** utility, see *z/OS Cryptographic Services System SSL Programming*. For information about using the **RACDCERT** command, see *z/OS Security Server RACF Command Language Reference*.

The user ID associated with the LDAP client must be authorized by RACF to use the PKCS #11 token. To authorize the LDAP client, you can use the RACF commands in the following example (where *NAME* is the name of the PKCS #11 token and *userid* is the user ID associated with the LDAP client utility).

```
SETROPTS CLASSACT(CRYPTOZ)
RDEFINE CRYPTOZ USER.NAME UACC(NONE)
RDEFINE CRYPTOZ SO.NAME UACC(NONE)
PERMIT USER.NAME CLASS(CRYPTOZ) ID(userid) ACCESS(READ)
PERMIT SO.NAME CLASS(CRYPTOZ) ID(userid) ACCESS(READ)
```

Remember to refresh RACF after doing the authorizations.

```
SETROPTS RACLIST(CRYPTOZ) REFRESH
```

After the PKCS #11 token is set up and authorized, specify the PKCS #11 token for the **-K** *keyFile* option using the following format:

```
-K *TOKEN*/NAME
```

Also, do not specify the **-P** *keyFilePW* option when using a PKCS #11 token.

SSL initialization failure

If SSL initialization fails, an error message like the following is returned:

```
ldap_ssl_client_init failed! rc == 113, failureReasonCode == 2
reason text: SSL initialization failed
```

The failureReasonCode indicates the cause of the SSL failure and is mapped from the return code of various SSL functions. See Table 7 for these values. The failure reason code and SSL return code mappings and #defines are documented in file **/usr/include/ldapssl.h**.

Table 7. SSL failure reason codes

Failure reason code	SSL return code	Failure reason code description
-1	402	No ciphers matched the server and client lists of acceptable ciphers
-2	403	No client certificate is to be used
-6	405	The certificate type is not supported
-10	406	I/O error communicating with peer application
-11	410	Incorrectly-formatted message received from peer application
-12	411	Message verification failed
-13	412	SSL protocol or certificate type is not supported

Table 7. SSL failure reason codes (continued)

Failure reason code	SSL return code	Failure reason code description
-14	413	Certificate signature is not correct for a certificate received from the peer
-15	414	Certificate is not valid
-16	415	Peer application has violated the SSL protocol
-17	416	Not authorized to access key database or SAF keyring
-18	417	Self-signed certificate cannot be validated
-20	4	Insufficient storage is available
-21	5	The environment or connection is not in the open state
-22	420	Socket closed by peer
-41	422	V3 cipher is not valid
-99	12 or any other unmapped SSL reason code	Unrecognized error
-1000	none	Failed loading SSL DLL
-1001	none	Failed locating SSL function
1	102	Keyring I/O error
2	202	Keyring open error
4	408	Keyring password is incorrect
12	6, 407	Keyfile label is not valid or certificate is not trusted
106	106	Key database file is corrupted
109	109	Key database or SAF key ring does not contain any valid CA certificates
201	201	Key database password or stash filename not set
203	203	Unable to generate temporary RSA key
204	204	Key database password is expired
301	301	Close failed
302	302	Connection has an active write
401	401	Validity time period for the certificate has expired
427	427	Unable to access the LDAP directory
428	428	The client key did not contain a private key
431	431	Certificate has been revoked
432	432	Session renegotiation is not allowed
433	433	Key exceeds allowable export size
434	434	Certificate key is not compatible with the negotiated cipher suite
435	435	Missing CA certificate
436	436	CRL cannot be processed
437	437	A close notification alert has been sent for the connection

Table 7. SSL failure reason codes (continued)

Failure reason code	SSL return code	Failure reason code description
438	438	Internal error reported by remote partner
439	439	Unknown alert received from remote partner
501	501	The buffer size is negative or zero
502	502	Operation would block
503	503	Read would be blocked
504	504	Write would be blocked
505	505	Record overflow
602	602	Function identifier is not valid
701	701	Attribute ID is not valid
702	702	Attribute length is not valid
703	703	Attribute enumeration value is not valid
705	705	Attribute value is not valid
706	706	Attribute parameter value is not valid
10001	1	Environment or SSL handle not valid
10003	3	Internal SSL error
10007	7	No certificate received from partner
10008	8	Certificate validation error
10009	9	Error processing cryptography
10010	10	Error validating ASN.1 fields in certificate
10011	11	Error connecting to LDAP server
10103	103	The database is not a key database

Using environment variables to control SSL/TLS settings

The z/OS LDAP client utilities do not support SSL V2 protocol, and disable it from being used. SSL V3, TLS V1.0, TLS V1.1, and TLS V1.2 protocols are supported. The z/OS System SSL defaults and environment variables control which of these supported protocols are enabled or disabled, and which cipher specifications apply. For example, the environment variable GSK_PROTOCOL_SSLV3 can be set to "ON" to enable SSL V3 protocol, or "OFF" to disable SSL V3 protocol. The environment variable GSK_PROTOCOL_TLSV1 can be set to "ON" to enable TLS V1.0 protocol, or "OFF" to disable TLS V1.0 protocol.

TLS V1.1 and TLS V1.2 protocols are disabled by default. To enable these protocol levels or to override the default cipher specifications, the z/OS System SSL environment variables can be used.

- Set `GSK_PROTOCOL_TLSV1_1 "ON"` to enable TLS V1.1 protocol.
- Set `GSK_PROTOCOL_TLSV1_2 "ON"` to enable TLS V1.2 protocol.
- Choose which cipher format is appropriate. Note that only one set of cipher suite specifications (2-byte or 4-byte) is applicable, depending on the setting of the `LDAP_SSL_CIPHER_FORMAT` environment variable.
  - If the default 2-byte cipher suites are sufficient, you can allow the settings to default. If you want to override the cipher suite specifications, and all can be specified as 2-byte cipher suite values, you can set `GSK_V3_CIPHER_SPECS` to

override the default cipher specifications, specifying the set of 2-byte values you want. You may also set the LDAP_SSL_CIPHER_FORMAT environment variable to CHAR2, or do not set it.

  - If you require any cipher suites that can only be specified as a 4-byte value, you must set GSK_V3_CIPHER_SPECS_EXPANDED to override the default cipher specifications, specifying the set of 4-byte values you want. You must also set the LDAP_SSL_CIPHER_FORMAT environment variable to CHAR4.

- Set GSK_SUITE_B_PROFILE to the value you want to apply Suite B-compliant options for your SSL connection. See *z/OS Cryptographic Services System SSL Programming* for more information. Note that enabling Suite B by using this environment variable causes the settings of the other environment variables noted above to be ignored, which includes the LDAP_SSL_CIPHER_FORMAT.

ldapchangepwd utility

Purpose

The **ldapchangepwd** utility provides an interface to the **ldap_modify()** API to allow the **userPassword** attribute value to be changed for the specified entry.

The **ldapchangepwd** utility opens a connection to an LDAP server, binds, and sends modify password requests to the LDAP server. The input consists of a distinguished name (DN), a current password value, and a new password value. The current password value is deleted from the **userPassword** attribute values for the specified distinguished name and is replaced with the new password value.

Format

```
ldapchangepwd -D bindDN -w currentpw -n newpw [options]
```

Parameters

options

Table 8 shows the *options* you can use for the **ldapchangepwd** utility:

Table 8. ldapchangepwd options

Option	Description
-?	Print this text.
-d *debugLevel*	Specify the level of debug messages to be created. The debug level is specified in the same fashion as the debug level for the LDAP server. See Table 5 on page 244 for the possible values for *debugLevel*. The default is no debug messages.
-D *bindDN*	Use *bindDN* to bind to the LDAP directory. The *bindDN* also identifies the DN whose **userPassword** attribute value is to be changed. The *bindDN* parameter is required and should be a string-represented DN. If the **-S** or **-m** option is equal to `DIGEST-MD5` or `CRAM-MD5`, this option is the authorization DN that is used for making access checks.
-g *realmName*	Specify the realm name to use when doing a `DIGEST-MD5` bind. This option is required when multiple realms are passed from an LDAP server to a client as part of a `DIGEST-MD5` challenge; otherwise, it is optional.
-h *ldapHost*	Specify the host name or IP address on which the LDAP server is running. The default is the local host.

Table 8. ldapchangepwd options (continued)

Option	Description
-K *keyFile*	Specify the name of the System SSL key database file, RACF key ring, or PKCS #11 token. If this option is not specified, this utility looks for the presence of the `SSL_KEYRING` environment variable with an associated name. If *keyFile* is specified as `*TOKEN*/NAME`, then System SSL uses the specified PKCS #11 token. Otherwise, System SSL uses a key database file or a RACF key ring. In this case, System SSL first assumes that *keyFile* is a key database file name and tries to locate the file. If *keyFile* is not a fully qualified z/OS UNIX System Services file name, the current directory is assumed to contain the key database file. The name cannot be a partitioned or sequential data set. If System SSL cannot locate the file, it then assumes that *keyFile* is a RACF key ring name. See "SSL/TLS information for LDAP client utilities" on page 259 for information about System SSL key databases, RACF key rings, and PKCS #11 tokens. This parameter is ignored if **-Z** is not specified.
-m *mechanism*	See the description of the **-S** option.
-M	Manage referral objects as normal entries. This requires a protocol level of 3.
-n *newpw*	Specify the new **userPassword** attribute value for the distinguished name (DN) specified in the **-D** option. This value replaces the current password specified in the **-w** option. Specify **?** to prompt for the new password value. This option is required.
-N *keyFileDN*	Specify the label associated with the certificate in the System SSL key database, RACF key ring, or PKCS #11 token. This parameter is ignored if **-Z** is not specified
-p *ldapPort*	Specify the TCP port where the LDAP server is listening. The default LDAP non-secure port is 389 and the default LDAP secure port is 636.
-P *keyFilePW*	Specify either the key database file password or the file specification for a System SSL password stash file. When the stash file is used, it must be in the form **file://** followed immediately (no blanks) by the file system file specification (for example, `file:///etc/ldap/sslstashfile`). The stash file must be a z/OS UNIX System Services file and cannot be a partitioned or sequential data set. This parameter is ignored if **-Z** is not specified.
-R	Do not automatically follow referrals.

Table 8. ldapchangepwd options (continued)

Option	Description
-S *mechanism* or **-m** *mechanism*	Specify the bind method to use. You can use either **-m** or **-S** to indicate the bind method. Specify `GSSAPI` to indicate a Kerberos Version 5 bind is requested, `EXTERNAL` to indicate that a certificate (SASL external) bind is requested, `CRAM-MD5` to indicate that a SASL Challenge Response Authentication Mechanism bind is requested, or `DIGEST-MD5` to indicate a SASL digest hash bind is requested. The `GSSAPI` method requires a protocol level of 3 and the user must have a valid Kerberos Ticket Granting Ticket in their credentials cache by using the Kerberos **kinit** command line utility. The `EXTERNAL` method requires a protocol level of 3. You must also specify **-Z**, **-K**, and **-P** to use certificate bind. If there is no default certificate in the key database file, RACF key ring, or PKCS #11 token or a certificate other than the default must be used, use the **-N** option to specify the label of the certificate. The `CRAM-MD5` method requires a protocol level of 3. The **-D** or **-U** option must be specified. The `DIGEST-MD5` method requires a protocol level of 3. The **-U** option must be specified. Optionally, the **-D** option can be used to specify the authorization DN. If **-m** or **-S** is not specified, a simple bind is performed.
-U *userName*	Specify the user name for `CRAM-MD5` or `DIGEST-MD5` binds. The *userName* is a short name (for example, the *uid* attribute value) that is used to perform bind authentication. This option is required if the **-S** or **-m** option is set to `DIGEST-MD5`.
-v	Use verbose mode, with many diagnostics written to standard output.
-V *version*	Specify the LDAP protocol level the client should use. The value for *version* can be **2** or **3**. The default is **3**.
-w *currentpw*	Use *currentpw* as the password for simple, `CRAM-MD5`, and `DIGEST-MD5` authentication. This value also specifies the current **userPassword** attribute value that is being changed for the distinguished name (DN) specified by the **-D** option. This value is replaced by the new password value specified in the **-n** option. Specify **?** to prompt for the current password value. This option is required.
-Z	Use a secure connection to communicate with the LDAP server. Secure connections expect the communication to begin with the SSL/TLS handshake. The **-K** *keyFile* option or equivalent environment variable is required when the **-Z** option is specified. The **-P** *keyFilePW* option is required when the **-Z** option is specified and the key file specifies a file system key database file. Unless you want to use the default certificate in the key database file, RACF key ring, or PKCS #11 token, use the **-N** option to specify the label of the certificate.

Examples

Examples of **ldapchangepwd** are:

- This example changes the **userPassword** attribute value for entry cn=jon,o=ibm,c=us from a1b2c3d4 to wxyz9876 is:

```
ldapchangepwd -D "cn=jon,o=ibm,c=us" -w a1b2c3d4 -n wxyz9876
```

- This example performs an EXTERNAL bind with the SSL client certificate, clientCert, in the /home/user/client.kdb SSL key database file. The authenticated user changes the **userPassword** value for entry cn=stanley,o=ibm,c=us from xyz123abc to abc321zyz:

```
ldapchangepwd -Z -m EXTERNAL -K /home/user/client.kdb -N clientCert -P secret
 -D "cn=stanley,o=ibm,c=us" -w xyz123abc -n abc321xyz
```

- This example performs a simple bind to prompt for the current and new **userPassword** attribute values for entry cn=yvonne,o=ibm,c=us. At the prompts, the user enters the current and new password values in a non-echoed manner for user cn=yvonne,o=ibm,c=us.

```
ldapchangepwd -D "cn=yvonne,o=ibm,c=us" -w ? -n ?
Enter current password ==>
Enter new password ==>
```

Notes

The LDAP_DEBUG environment variable can be used to set the debug level. For more information about specifying the debug level using keywords, decimal, hexadecimal, and plus and minus syntax, see “Enabling tracing” on page 242.

You can specify an LDAP URL for *ldapHost* on the **-h** parameter. See “ldap_init()” on page 107 for more information.

For information about SSL/TLS, see “SSL/TLS information for LDAP client utilities” on page 259.

The authenticating user must have the appropriate permissions to update the **userPassword** attribute for the distinguished name specified in the **-D** option.

The password prompt (-w ?) is not supported when running from TSO or batch. In these environments, the password value must be specified on the **-w** option.

The **getpass()** routine used to prompt for the password returns at most PASS_MAX number of characters, truncating any additional characters. See the description of **getpass()** in *z/OS XL C/C++ Runtime Library Reference* for more information. If the length of the specified password is greater than PASS_MAX, the password value must be specified on the **-w** option.

Diagnostics

Exit status is 0 if no errors occur. Errors result in a nonzero exit status and a diagnostic message being written to standard error. If the errors are caused by the password policy requirements not being met, the **Effective password policy** extended operation is invoked and the effective password policy entries and attributes values are written to standard output.

ldapcompare utility

Purpose

The **ldapcompare** utility provides an interface to the **ldap_compare()** API.

The **ldapcompare** utility opens a connection to an LDAP server, binds, and does one or more compares for an attribute value in an entry. The input consists of a distinguished name (DN) and an attribute type and value to compare. For each set of input, a comparison is performed for the specified attribute in the entry with that DN. If the DN and attribute type and value are not provided, the input is read from standard input or from *file* if the **-f** option is used, and two lines of input are read for each comparison. The first line contains the DN and the second line contains the attribute type and value.

Format

```
ldapcompare [options] [dn attr=value]...
```

Parameters

options

Table 9 shows the *options* you can use for the **ldapcompare** utility:

Table 9. ldapcompare options

Option	Description
-?	Print this text.
-c	Continuous operation mode. Errors are reported, but **ldapcompare** continues with comparisons. The return code from the utility is determined by the last comparison. The default is to exit after reporting an error.
-d *debugLevel*	Specify the level of debug messages to be created. The debug level is specified in the same fashion as the debug level for the LDAP server. See Table 5 on page 244 for the possible values for *debugLevel*. The default is no debug messages.
-D *bindDN*	Use *bindDN* to bind to the LDAP directory. The *bindDN* parameter should be a string-represented DN. The default is a `NULL` string. If the **-S** or **-m** option is equal to `DIGEST-MD5` or `CRAM-MD5`, this option is the authorization DN that is used for making access checks. This directive is optional when used in this manner.
-f *file*	Read the compare input from *file* instead of from standard input or the command line (by specifying *dn* and *attr=value*). An LDAP compare is performed for every set of two lines in the file. The first line in the set specifies the DN of the entry to compare. The second line contains the *attr=value* specification, indicating the attribute and value to compare. Do not put double quotation marks around the DN or attribute values in the file. You can specify a partitioned or sequential data set for *file* on the **-f** parameter. See "Specifying a value for a file name" on page 258 for more information.
-g *realmName*	Specify the realm name to use when doing a `DIGEST-MD5` bind. This option is required when multiple realms are passed from an LDAP server to a client as part of a `DIGEST-MD5` challenge; otherwise, it is optional.

Table 9. ldapcompare options (continued)

Option	Description
-h *ldapHost*	Specify the host name or IP address on which the LDAP server is running. The default is the local host.
-K *keyFile*	Specify the name of the System SSL key database file, RACF key ring, or PKCS #11 token. If this option is not specified, this utility looks for the presence of the `SSL_KEYRING` environment variable with an associated name. If *keyFile* is specified as `*TOKEN*/`*`NAME`*, then System SSL uses the specified PKCS #11 token. Otherwise, System SSL uses a key database file or a RACF key ring. In this case, System SSL first assumes that *keyFile* is a key database file name and tries to locate the file. If *keyFile* is not a fully-qualified z/OS UNIX System Services file name, the current directory is assumed to contain the key database file. The name cannot be a partitioned or sequential data set. If System SSL cannot locate the file, it then assumes that *keyFile* is a RACF key ring name. See "SSL/TLS information for LDAP client utilities" on page 259 for information about System SSL key databases, RACF key rings, and PKCS #11 tokens. This parameter is ignored if **-Z** is not specified.
-m *mechanism*	See the description of the **-S** option.
-M	Manage referral objects as normal entries. This requires a protocol level of 3.
-n	Show what would be done, but do not actually compare entries. Useful for debugging with **-v**.
-N *keyFileDN*	Specify the label associated with the certificate in the System SSL key database, RACF key ring, or PKCS #11 token. This parameter is ignored if **-Z** is not specified
-p *ldapPort*	Specify the TCP port where the LDAP server is listening. The default LDAP non-secure port is 389 and the default LDAP secure port is 636.
-P *keyFilePW*	Specify either the key database file password or the file specification for a System SSL password stash file. When the stash file is used, it must be in the form **`file://`** followed immediately (no blanks) by the file system file specification (for example, `file:///etc/ldap/sslstashfile`). The stash file must be a z/OS UNIX System Services file and cannot be a partitioned or sequential data set. This parameter is ignored if **-Z** is not specified.
-R	Do not automatically follow referrals.

Table 9. ldapcompare options (continued)

Option	Description
-S *mechanism* or **-m** *mechanism*	Specify the bind method to use. You can use either **-m** or **-S** to indicate the bind method. Specify `GSSAPI` to indicate a Kerberos Version 5 bind is requested, `EXTERNAL` to indicate that a certificate (SASL external) bind is requested, `CRAM-MD5` to indicate that a SASL Challenge Response Authentication Mechanism bind is requested, or `DIGEST-MD5` to indicate a SASL digest hash bind is requested. The `GSSAPI` method requires a protocol level of 3 and the user must have a valid Kerberos Ticket Granting Ticket in their credentials cache by using the Kerberos **kinit** command line utility. The `EXTERNAL` method requires a protocol level of 3. You must also specify **-Z**, **-K**, and **-P** to use certificate bind. If there is no default certificate in the key database file, RACF key ring, or PKCS #11 token or a certificate other than the default must be used, use the **-N** option to specify the label of the certificate. The `CRAM-MD5` method requires a protocol level of 3. The **-D** or **-U** option must be specified. The `DIGEST-MD5` method requires a protocol level of 3. The **-U** option must be specified. Optionally, the **-D** option can be used to specify the authorization DN. If **-m** or **-S** is not specified, a simple bind is performed.
-U *userName*	Specify the user name for `CRAM-MD5` or `DIGEST-MD5` binds. The *userName* is a short name (for example, the *uid* attribute value) that is used to perform bind authentication. This option is required if the **-S** or **-m** option is set to `DIGEST-MD5`.
-v	Use verbose mode, with many diagnostics written to standard output.
-V *version*	Specify the LDAP protocol level the client should use. The value for *version* can be **2** or **3**. The default is **3**.
-w *passwd*	Use *passwd* as the password for simple, `CRAM-MD5`, and `DIGEST-MD5` authentication. The default is a `NULL` string.
-Z	Use a secure connection to communicate with the LDAP server. Secure connections expect the communication to begin with the SSL/TLS handshake. The **-K** *keyFile* option or equivalent environment variable is required when the **-Z** option is specified. The **-P** *keyFilePW* option is required when the **-Z** option is specified and the key file specifies a file system key database file. Unless you want to use the default certificate in the key database file, RACF key ring, or PKCS #11 token, use the **-N** option to specify the label of the certificate.

dn Specify the DN of the entry to compare.

attr=value
Specify the attribute type and the value to compare. An error is returned if the entry does not contain the attribute to be compared.

All other command line inputs result in a syntax error message, after which the correct syntax is displayed. If the same option is specified multiple times or if both

-m and **-S** are specified, the last value specified is used.

Examples

Following are some **ldapcompare** examples:

- The following command compares the `sn` attribute within the entry named `cn=Compare Me, o=My Company, c=US`. The command returns true if the `sn` attribute value is `Smith` and false if it is not.

  ```
  ldapcompare "cn=Compare Me, o=My Company, c=US" sn=Smith
  ```

- The following example uses file input to compare the `telephonenumber` attribute within the entry named `cn=ken, o=My Company, c=US` and to compare the description attribute within the entry named `cn=jay, o=My Company, c=US`. A separate result is returned for each comparison. Assume that `/tmp/compareFile` contains:

  ```
  cn=ken, o=My Company, c=US
  telephonenumber=123-456-7890

  cn=jay, o=My Company, c=US
  description=LDAP development
  ```

  The following command performs the comparisons:

  ```
  ldapcompare -f /tmp/compareFile
  ```

- For z/OS LDAP support for RACF access, the following command determines if the OMVS UID of RACF user `u1` is 123. It is assumed that the z/OS LDAP support for RACF access suffix is `sysplex=sysplexa`.

  ```
  ldapcompare -D racfid=admin1,profiletype=user,sysplex=sysplexa -w passwd
      "racfid=u1,profiletype=user,sysplex=sysplexa" racfomvsuid=123
  ```

Notes

If no *dn* and *attr=value* arguments are provided and the **-f** option is not used, the **ldapcompare** command waits to read a list of DNs and attribute types and values from standard input. To break out of the wait, press the Ctrl+C keys or the Ctrl+D keys.

The `LDAP_DEBUG` environment variable can be used to set the debug level. For more information about specifying the debug level using keywords, decimal, hexadecimal, and plus and minus syntax, see “Enabling tracing” on page 242.

You can specify an LDAP URL for *ldapHost* on the **-h** parameter. See “ldap_init()” on page 107 for more information.

For information about SSL/TLS, see “SSL/TLS information for LDAP client utilities” on page 259.

Diagnostics

Exit status is `5` (`LDAP_COMPARE_FALSE`) or `6` (`LDAP_COMPARE_TRUE`) if no errors occur. Errors result in a nonzero exit status and a diagnostic message being written to standard error.

ldapdelete utility

Purpose

The **ldapdelete** utility provides an interface to the **ldap_delete()** API.

The **ldapdelete** utility opens a connection to an LDAP server, binds, and deletes one or more entries. If one or more *dn* arguments are provided, entries with those DNs are deleted. If no *dn* arguments are provided, the input is read from standard input or from *file* if the **-f** option is used. Each line of input contains the DN of an entry to be deleted. Each entry to be deleted must be a *leaf* entry (an entry with no subordinate entries) or it must become a leaf entry when the previously specified entries are deleted.

Format

```
ldapdelete [options] [dn]...
```

Parameters

options

Table 10 shows the *options* you can use for the **ldapdelete** utility:

Table 10. ldapdelete options

Option	Description
-?	Print this text.
-c	Continuous operation mode. Errors are reported, but **ldapdelete** continues with deletions. The return code from the utility is determined by the last deletion. The default is to exit after reporting an error.
-d *debugLevel*	Specify the level of debug messages to be created. The debug level is specified in the same fashion as the debug level for the LDAP server. See Table 5 on page 244 for the possible values for *debugLevel*. The default is no debug messages.
-D *bindDN*	Use *bindDN* to bind to the LDAP directory. The *bindDN* parameter should be a string-represented DN. The default is a `NULL` string. If the **-S** or **-m** option is equal to `DIGEST-MD5` or `CRAM-MD5`, this option is the authorization DN that is used for making access checks. This directive is optional when used in this manner.
-f *file*	Read a series of lines from *file*, performing one LDAP delete for the DN on each line. Do not put quotation marks around the DN values in the file. You can specify a partitioned or sequential data set for *file* on the **-f** parameter. See "Specifying a value for a file name" on page 258 for more information.
-g *realmName*	Specify the realm name to use when doing a `DIGEST-MD5` bind. This option is required when multiple realms are passed from an LDAP server to a client as part of a `DIGEST-MD5` challenge; otherwise, it is optional.
-h *ldapHost*	Specify the host name or IP address on which the LDAP server is running. The default is the local host.

Table 10. ldapdelete options (continued)

Option	Description
-k	Send the Server Administration control with the operation request. The control requires a protocol level of 3 and its criticality is set to `TRUE`. There is no control value. This control enables a server that would normally refuse updates, such as a quiesced or replica server, to allow updates. See *z/OS IBM Tivoli Directory Server Administration and Use for z/OS* for additional information about this control.
-K *keyFile*	Specify the name of the System SSL key database file, RACF key ring, or PKCS #11 token. If this option is not specified, this utility looks for the presence of the `SSL_KEYRING` environment variable with an associated name. If *keyFile* is specified as `*TOKEN*/`*`NAME`*, then System SSL uses the specified PKCS #11 token. Otherwise, System SSL uses a key database file or a RACF key ring. In this case, System SSL first assumes that *keyFile* is a key database file name and tries to locate the file. If *keyFile* is not a fully-qualified z/OS UNIX System Services file name, the current directory is assumed to contain the key database file. The name cannot be a partitioned or sequential data set. If System SSL cannot locate the file, it then assumes that *keyFile* is a RACF key ring name. See "SSL/TLS information for LDAP client utilities" on page 259 for information about System SSL key databases, RACF key rings, and PKCS #11 tokens. This parameter is ignored if **-Z** is not specified.
-L	Send the Do Not Replicate control with the operation request. The control requires a protocol level of 3 and its criticality is set to `TRUE`. There is no control value. This control prevents the targeted server from sending replicated entries to the next tier of advanced replication servers. See *z/OS IBM Tivoli Directory Server Administration and Use for z/OS* for additional information about this control.
-m *mechanism*	See the description of the **-S** option.
-M	Manage referral objects as normal entries. This requires a protocol level of 3.
-n	Show what would be done, but do not actually delete entries. Useful for debugging with **-v**.
-N *keyFileDN*	Specify the label associated with the certificate in the System SSL key database, RACF key ring, or PKCS #11 token. This parameter is ignored if **-Z** is not specified
-p *ldapPort*	Specify the TCP port where the LDAP server is listening. The default LDAP non-secure port is 389 and the default LDAP secure port is 636.
-P *keyFilePW*	Specify either the key database file password or the file specification for a System SSL password stash file. When the stash file is used, it must be in the form **`file://`** followed immediately (no blanks) by the file system file specification (for example, `file:///etc/ldap/sslstashfile`). The stash file must be a z/OS UNIX System Services file and cannot be a partitioned or sequential data set. This parameter is ignored if **-Z** is not specified.
-R	Do not automatically follow referrals.

Table 10. ldapdelete options (continued)

Option	Description
-S *mechanism* or **-m** *mechanism*	Specify the bind method to use. You can use either **-m** or **-S** to indicate the bind method. Specify GSSAPI to indicate a Kerberos Version 5 bind is requested, EXTERNAL to indicate that a certificate (SASL external) bind is requested, CRAM-MD5 to indicate that a SASL Challenge Response Authentication Mechanism bind is requested, or DIGEST-MD5 to indicate a SASL digest hash bind is requested. The GSSAPI method requires a protocol level of 3 and the user must have a valid Kerberos Ticket Granting Ticket in their credentials cache by using the Kerberos **kinit** command line utility. The EXTERNAL method requires a protocol level of 3. You must also specify **-Z**, **-K**, and **-P** to use certificate bind. If there is no default certificate in the key database file, RACF key ring, or PKCS #11 token or a certificate other than the default must be used, use the **-N** option to specify the label of the certificate. The CRAM-MD5 method requires a protocol level of 3. The **-D** or **-U** option must be specified. The DIGEST-MD5 method requires a protocol level of 3. The **-U** option must be specified. Optionally, the **-D** option can be used to specify the authorization DN. If **-m** or **-S** is not specified, a simple bind is performed.
-U *userName*	Specify the user name for CRAM-MD5 or DIGEST-MD5 binds. The *userName* is a short name (for example, the *uid* attribute value) that is used to perform bind authentication. This option is required if the **-S** or **-m** option is set to DIGEST-MD5.
-v	Use verbose mode, with many diagnostics written to standard output.
-V *version*	Specify the LDAP protocol level the client should use. The value for *version* can be **2** or **3**. The default is **3**.
-w *passwd*	Use *passwd* as the password for simple, CRAM-MD5, and DIGEST-MD5 authentication. The default is a NULL string.
-Z	Use a secure connection to communicate with the LDAP server. Secure connections expect the communication to begin with the SSL/TLS handshake. The **-K** *keyFile* option or equivalent environment variable is required when the **-Z** option is specified. The **-P** *keyFilePW* option is required when the **-Z** option is specified and the key file specifies a file system key database file. Unless you want to use the default certificate in the key database file, RACF key ring, or PKCS #11 token, use the **-N** option to specify the label of the certificate.

dn Specify distinguished name (DN) of an entry to delete. You can specify one or more *dn* arguments. Each *dn* should be a string-represented DN.

All other command line inputs result in a syntax error message, after which the correct syntax is displayed. If the same option is specified multiple times or if both **-m** and **-S** are specified, the last value specified is used.

Examples

Following are some **ldapdelete** examples:

- The following command attempts to delete the entry named with `commonName Delete Me` directly below `My Company` organizational entry. It might be necessary to supply a *bindDN* and *passwd* for deletion to be allowed. (See the **-D** and **-w** options.)

  ```
  ldapdelete "cn=Delete Me, o=My Company, c=US"
  ```

- The following example uses file input to delete the `cn=ken, o=My Company, c=US` and `cn=jay, o=My Company, c=US` entries. Assume that `/tmp/deleteFile` contains:

  ```
  cn=ken, o=My Company, c=US
  cn=jay, o=My Company, c=US
  ```

  The following command performs the deletions:

  ```
  ldapdelete -f /tmp/deleteFile
  ```

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

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

Notes

If no *dn* arguments are provided and the **-f** option is not specified, the **ldapdelete** command waits to read a list of DNs from standard input. To break out of the wait, press the Ctrl+C keys or the Ctrl+D keys.

The `LDAP_DEBUG` environment variable can be used to set the debug level. For more information about specifying the debug level using keywords, decimal, hexadecimal, and plus and minus syntax, see “Enabling tracing” on page 242.

You can specify an LDAP URL for *ldapHost* on the **-h** parameter. See “ldap_init()” on page 107 for more information.

For information about SSL/TLS, see “SSL/TLS information for LDAP client utilities” on page 259.

Diagnostics

Exit status is `0` if no errors occur. Errors result in a nonzero exit status and a diagnostic message being written to standard error.

ldapmodify and ldapadd utilities

Purpose

The **ldapmodify** utility provides an interface to the **ldap_modify()** and **ldap_add()** APIs. The **ldapadd** command is implemented as a renamed version of **ldapmodify**. When invoked as **ldapadd**, the **-a** (add new entry) flag is turned on automatically.

The **ldapmodify** utility opens a connection to an LDAP server, binds, and modifies or adds entries. The entry information is read from standard input (or an input file that is redirected to standard input) or from *file* by using the **-f** option.

Format

```
ldapmodify | ldapadd [options]
```

Parameters

options

Table 11 shows the options that you can use for the **ldapmodify** and **ldapadd** utilities:

Table 11. ldapmodify and ldapadd options

Option	Description
-?	Print this text.
-a	Add new entries. The default for **ldapmodify** is to modify existing entries. If invoked as **ldapadd**, this flag is always set.
-b	Assume that any attribute values that start with a slash (`/`) are binary values that are contained in a file. The location of the binary file must be specified as a fully qualified z/OS UNIX System Services file system or as a fully qualified data set name. If a data set name is specified, it must start with two slashes (`//`) and the name must have single quotation marks (`'`) around it.
-c	Continuous operation mode. Errors are reported, but **ldapmodify** continues with modifications. The return code from the utility is determined by the last modification. The default is to exit after reporting an error.
-d *debugLevel*	Specify the level of debug messages to be created. The debug level is specified in the same fashion as the debug level for the LDAP server. See Table 5 on page 244 for the possible decimal values for *debugLevel*. The default is no debug messages.
-D *bindDN*	Use *bindDN* to bind to the LDAP directory. The *bindDN* parameter should be a string-represented DN. The default is a `NULL` string. If the **-S** or **-m** option is equal to `DIGEST-MD5` or `CRAM-MD5`, this option is the authorization DN that is used for making access checks. This directive is optional when used in this manner.
-f *file*	Read the entry modification information from *file* instead of from standard input. You can specify a partitioned or sequential data set for *file* on the **-f** parameter. See "Specifying a value for a file name" on page 258 for more information.
-F	Force application of all changes regardless of the contents of input lines that begin with **`replica:`** (by default, **`replica:`** lines are compared against the LDAP server host and port in use to decide if a replication log record should be applied).

Table 11. ldapmodify and ldapadd options (continued)

Option	Description
-g *realmName*	Specify the realm name to use when doing a DIGEST-MD5 bind. This option is required when multiple realms are passed from an LDAP server to a client as part of a DIGEST-MD5 challenge; otherwise, it is optional.
-h *ldapHost*	Specify the host name or IP address on which the LDAP server is running. The default is the local host.
-k	Send the Server Administration control with the operation request. The control requires a protocol level of 3 and its criticality is set to `TRUE`. There is no control value. This control enables a server that would normally refuse updates, such as a quiesced or replica server, to allow updates. See *z/OS IBM Tivoli Directory Server Administration and Use for z/OS* for additional information about this control.
-K *keyFile*	Specify the name of the System SSL key database file, RACF key ring, or PKCS #11 token. If this option is not specified, this utility looks for the presence of the `SSL_KEYRING` environment variable with an associated name. If *keyFile* is specified as `*TOKEN*/`*`NAME`*, then System SSL uses the specified PKCS #11 token. Otherwise, System SSL uses a key database file or a RACF key ring. In this case, System SSL first assumes that *keyFile* is a key database file name and tries to locate the file. If *keyFile* is not a fully qualified z/OS UNIX System Services file name, the current directory is assumed to contain the key database file. The name cannot be a partitioned or sequential data set. If System SSL cannot locate the file, it then assumes that *keyFile* is a RACF key ring name. See "SSL/TLS information for LDAP client utilities" on page 259 for information about System SSL key databases, RACF key rings, and PKCS #11 tokens. This parameter is ignored if **-Z** is not specified.
-L	Send the Do Not Replicate control with the operation request. The control requires a protocol level of 3 and its criticality is set to `TRUE`. There is no control value. This control prevents the targeted server from sending replicated entries to the next tier of advanced replication servers. See *z/OS IBM Tivoli Directory Server Administration and Use for z/OS* for additional information about this control.
-m *mechanism*	See the description of the **-S** option.
-M	Manage referral objects as normal entries. This requires a protocol level of 3.
-n	Show what would be done, but do not actually modify entries. Useful for debugging with **-v**.
-N *keyFileDN*	Specify the label associated with the certificate in the System SSL key database, RACF key ring, or PKCS #11 token. This parameter is ignored if **-Z** is not specified.
-p *ldapPort*	Specify the TCP port where the LDAP server is listening. The default LDAP non-secure port is 389 and the default LDAP secure port is 636.

Table 11. ldapmodify and ldapadd options (continued)

Option	Description
-P *keyFilePW*	Specify either the key database file password or the file specification for a System SSL password stash file. When the stash file is used, it must be in the form **file://** followed immediately (no blanks) by the file system file specification (for example, `file:///etc/ldap/sslstashfile`). The stash file must be a z/OS UNIX System Services file and cannot be a partitioned or sequential data set. This parameter is ignored if **-Z** is not specified.
-r	Replace existing values by default.
-R	Do not automatically follow referrals.
-S *mechanism* or **-m** *mechanism*	Specify the bind method to use. You can use either **-m** or **-S** to indicate the bind method. Specify `GSSAPI` to indicate that a Kerberos Version 5 bind is requested, `EXTERNAL` to indicate that a certificate (SASL external) bind is requested, `CRAM-MD5` to indicate that a SASL Challenge Response Authentication Mechanism bind is requested, or `DIGEST-MD5` to indicate that a SASL digest hash bind is requested. The `GSSAPI` method requires a protocol level of 3 and the user must have a valid Kerberos Ticket Granting Ticket in their credentials cache by using the Kerberos **kinit** command-line utility. The `EXTERNAL` method requires a protocol level of 3. You must also specify **-Z**, **-K**, and **-P** to use certificate bind. If there is no default certificate in the key database file, RACF key ring, or PKCS #11 token or a certificate other than the default must be used, use the **-N** option to specify the label of the certificate. The `CRAM-MD5` method requires a protocol level of 3. The **-D** or **-U** option must be specified. The `DIGEST-MD5` method requires a protocol level of 3. The **-U** option must be specified. Optionally, the **-D** option can be used to specify the authorization DN. If **-m** or **-S** is not specified, a simple bind is performed.
-u on | off	Specify whether the **ldapmodify** utility sends the `SchemaReplaceByValueControl` control to the server. This control indicates how a schema modify reacts to a replace modification. If set to `off`, a schema modification removes all current values for an attribute and replaces them with the set of values in the replace operation. If set to `on`, an attribute value is updated if some value in the replace operation has the same numeric object identifier as a value that exists in the schema attribute. An attribute value is added if the numeric object identifier in the replace operation does not exist in the schema attribute. No attribute values are removed.
-U *userName*	Specify the user name for CRAM-MD5 or DIGEST-MD5 binds. The *userName* is a short name (for example, the *uid* attribute value) that is used to perform bind authentication. This option is required if the **-S** or **-m** option is set to `DIGEST-MD5`.
-v	Use verbose mode, with many diagnostics written to standard output.
-V *version*	Specify the LDAP protocol level the client should use. The value for *version* can be `2` or `3`. The default is `3`.

Table 11. ldapmodify and ldapadd options (continued)

Option	Description
-w *passwd*	Use *passwd* as the password for simple, `CRAM-MD5`, and `DIGEST-MD5` authentication. The default is a `NULL` string.
-Z	Use a secure connection to communicate with the LDAP server. Secure connections expect the communication to begin with the SSL/TLS handshake. The **-K** *keyFile* option or equivalent environment variable is required when the **-Z** option is specified. The **-P** *keyFilePW* option is required when the **-Z** option is specified and the key file specifies a file system key database file. Unless you want to use the default certificate in the key database file, RACF key ring, or PKCS #11 token, use the **-N** option to specify the label of the certificate.

All other command-line inputs result in a syntax error message, after which the correct syntax is displayed. If the same option is specified multiple times or if both **-m** and **-S** are specified, the last value specified is used.

LDAP Data Interchange Format (LDIF)

LDAP Data Interchange Format (LDIF) is a standard text format for representing LDAP objects and LDAP updates (add, modify, delete, modify DN). Files containing LDIF records are used to transfer data between directory servers or used as input by LDAP utilities such as **ldapadd** and **ldapmodify**.

LDIF content records are used to represent LDAP directory content and consist of a line identifying the object, followed by optional lines containing controls, which are then followed by lines containing the attribute-value pairs for the object. This type of file is used by the **ldapadd, ds2ldif**, and **ldif2ds** utilities. See ds2ldif utility and ldif2ds utility in *z/OS IBM Tivoli Directory Server Administration and Use for z/OS* for more information about the **ds2ldif** and **ldif2ds** utilities.

LDIF change records are used to represent directory updates. These records consist of a line identifying the directory object, followed by lines describing the changes to the object. The changes include adding, deleting, renaming, or moving objects, and modifying existing objects.

The input styles for content and change records are:

- A standard LDIF style that is defined by RFC 2849: *The LDAP Data Interchange Format (LDIF)*
- A non-standard "modify style"

Use of the standard LDIF style is suggested; the non-standard style is documented later for use with older tools that produce or use that style.

Input styles

The **ldapmodify** and **ldapadd** commands accept two forms of input. The type of input is determined by the format of the first input line supplied to **ldapmodify** or **ldapadd**.

The first line of input to the **ldapmodify** or **ldapadd** command must denote the distinguished name of a directory entry to add or modify. This input line must be of the form:

```
dn:distinguished_name

or

distinguished_name
```

where **dn:** is a literal string and *distinguished_name* is the distinguished name of the directory entry to modify (or add). If **dn:** is found, the input style is set to RFC 2849 LDIF style. If it is not found, the input style is set to "modify style".

Note:

1. The **ldapadd** command is equivalent to invoking the **ldapmodify -a** command.
2. The **ldapmodify** and **ldapadd** utilities do not support base64 encoded distinguished names.

RFC 2849 LDIF input

When using RFC 2849 LDIF input, attribute types and values are delimited by a single colon (**:**) or a double colon (**::**). Furthermore, individual changes to attribute values are delimited with a **changetype:** input line. The general form of input lines for RFC 2849 LDIF is:

```
change_record
<blank line>
change_record
<blank line>
⋮
```

In RFC 2849 LDIF input:

1. A comment line is a line that begins with a number sign (#) in column 1. Comment lines are ignored.
2. A continuation line is a line that begins with a space in column 1. The rest of the continuation line, starting in column 2, is appended to the previous line.
3. Ensure that there are no extraneous spaces or characters at the end of a line. Even if not viewable in an editor, these characters are part of the modify input and can produce unexpected errors or unusable data.

An input file in RFC 2849 LDIF style consists of one or more *change_record* sets of lines that are separated by one or more blank lines. Each *change_record* has the following form:

```
dn:distinguished_name
[control:control_oid[true|false][::control_value]]
[changetype:{modify|add|modrdn|delete}]
{change_clause
⋮
}
```

A *change_record* consists of a line indicating the distinguished name of the entry directory to be modified, one or more optional lines indicating controls to be sent to the server on the modification, an optional line indicating the type of modification to be performed against the directory entry, and one or more *change_clause* sets of lines.

If one or more `control:` lines are present, the *control_oid* indicates the OID of the control, `true` or `false` may optionally be specified to indicate the criticality of the control (defaults to `false` if not specified), and an optional *control_value* can be specified. The *control_value* is expected to be in base64 format. This format is an

encoding that represents every three binary bytes with four text characters. See the **base64encode()** function in `/usr/lpp/ldap/examples/line64.c` for an implementation of base64 encoding.

The control lines in the RFC 2849 LDIF input style provide a way to apply certain controls to an individual entry rather than using a command-line option. For example, the **-M, -L**, and **-k** command-line options send the controls that you want for all entries in the RFC 2849 LDIF input style. Any acceptable server control can be specified on the control lines. If the same control is specified multiple times for an entry, the client sends multiple controls for the entry to the server. This can occur if the control is specified using a command-line option and on a control line for the entry, or on multiple control lines for the entry.

If the **changetype:** line is omitted, the change type is assumed to be **modify** unless the command invocation was **ldapmodify -a** or **ldapadd**, in which case the **changetype** is assumed to be **add**.

When the change type is **modify**, each *change_clause* is defined as a set of lines of the form:

add:x
{*attrtype*}{*sep*}{*value*}
⋮
-

or

replace:x
{*attrtype*}{*sep*}{*value*}
⋮
-

or

delete:{*attrtype*}
[{*attrtype*}{*sep*}{*value*}]
⋮
-

or

{*attrtype*}{*sep*}{*value*}
⋮

Specifying **replace** replaces all existing values for the attribute with the specified set of attribute values except when modifying a schema entry by using the **-u** option with `SchemaReplaceByValueControl` enabled. (See the description of the **-u** option in Table 11 on page 276.) Specifying **add** adds to the existing set of attribute values. Specifying **delete** without any attribute-value pair records removes all the values for the specified attribute. Specifying **delete** followed by one or more attribute-value pair records removes only those values specified in the attribute-value pair records.

If an **add:x**, **replace:x**, or **delete:***attrtype* line (a change indicator) is specified, a line containing a hyphen (**-**) is expected as a closing delimiter for the changes. Attribute-value pairs are expected on the input lines that are found between the change indicator and hyphen line. If the change indicator line is omitted, the change is assumed to be **add** for the attribute values specified. However, if the **-r** option is specified on **ldapmodify**, the *change_clause* is assumed to be **replace**. The separator, *sep*, can be either a single colon (**:**) or a double colon (**::**). A single colon

(**:**) is used as a separator when *value* contains printable characters while a double colon (**::**) is used as a separator when *value* contains non-printable characters or begins with a space. Any white space characters between the separator and the attribute value are ignored. If a double colon is used as the separator, the input is expected to be in base64-encoded format. This format is an encoding that represents every three binary bytes with four text characters. See the **base64encode()** function in `/usr/lpp/ldap/examples/line64.c` for an implementation of this encoding.

Multiple attribute values are specified by using multiple {*attrtype*}{*sep*}{*value*} specifications.

Note: RFC 2849 indicates that LDIF file parsers should support the `file://` URL format on a value to indicate that the contents of the referenced file are to be included verbatim in the integrated input of the LDIF file. However, the z/OS LDAP client LDIF parser does not support this specification. Also, the z/OS LDAP client LDIF parser does not support language or syntax tags on *attrtype*.

When the change type is **add**, each *change_clause* is defined as a set of lines of the form:

{*attrtype*}{*sep*}{*value*}

As with change type of **modify**, the separator, *sep*, can be either a single colon (**:**) or a double colon (**::**). A single colon (**:**) is used as a separator when *value* contains printable characters while a double colon (**::**) is used as a separator when *value* contains non-printable characters or begins with a space. Any white space characters between the separator and the attribute value are ignored. Attribute values can be continued across multiple lines by using a single space character as the first character of the next line of input. If a double colon is used as the separator, the input is expected to be in base64-encoded format.

When the change type is **modrdn**, each *change_clause* is defined as a set of lines of the form:

newrdn:*value*
deleteoldrdn:{**0**|**1**}

These are the parameters that you can specify on a modify RDN LDAP operation. The value for the **newrdn** setting is the new RDN to be used when performing the modify RDN operation. Specify `0` for the value of the **deleteoldrdn** setting to save the attribute in the old RDN and specify `1` to remove the attribute values in the old RDN. You cannot use **ldapmodify** to move an entry under a new superior DN, instead, use "ldapmodrdn utility" on page 291 with the **-s** option.

When the change type is **delete**, no *change_clause* is specified.

RFC 2849 LDIF style examples

Here are some examples of valid input for the **ldapmodify** command using RFC 2849 LDIF style.

Adding a new entry

The following example adds a new entry into the directory using name `cn=Tim Doe, ou=Your Department, o=Your Company, c=US`, assuming **ldapadd** or **ldapmodify -a** is invoked:

```
dn:cn=Tim Doe, ou=Your Department, o=Your Company, c=US
changetype:add
cn: Tim Doe
sn: Doe
objectclass: organizationalperson
objectclass: person
objectclass: top
```

The following example sends the Server Administration Control (OID 1.3.18.0.2.10.15) and the Do Not Replicate Control (OID 1.3.18.0.2.10.23) and adds a new entry into the directory by using name cn=Tim Doe, ou=Your Department, o=Your Company, c=US, assuming **ldapadd** or **ldapmodify -a** is invoked:

```
dn:cn=Tim Doe, ou=Your Department, o=Your Company, c=US
control: 1.3.18.0.2.10.15
control: 1.3.18.0.2.10.23
changetype:add
cn: Tim Doe
sn: Doe
objectclass: organizationalperson
objectclass: person
objectclass: top
```

The following example adds a new entry that contains a binary user certificate that is base64-encoded in the userCertificate attribute into the directory by using name cn=John Doe, ou=Your Department, o=Your Company, c=US, assuming **ldapadd** or **ldapmodify -a** is invoked:

```
dn: cn=John Doe, ou=Your Department, o=Your Company, c=US
changetype:add
cn: John Doe
sn: Doe
usercertificate:: MIICNjCCAZ+gAwIBAgIBADANBgkqhkiG9w0BAQUFADAvMQswCQYDVQQGEwJ
 1czEMMAoGA1UEChMDaWJtMRIwEAYDVQQDEwlyMTNzZXJ2ZXIwHhcNMTAwNjE1MDQwMDAwWhcNMjE
 wMTAxMDM1OTU5WjAvMQswCQYDVQQGEwJ1czEMMAoGA1UEChMDaWJtMRIwEAYDVQQDEwlyMTNzZXJ
 2ZXIwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMVgV8f3IAZEZ5/h3R2Iy7h4LSHbhsj4diH
 iHPIpTRtqJD5d42z2Z4gG9oUzqfYLyZSPoAVlDwVbufZVVvBeiDo7Bgm+1nj4/YYWCpnCkETmriB
 bVDJBoaF8W9xxHs38F6LVuJniDMp0VT9lDcqH3RNWgIcDqKurQm2uTHNDs6OtAgMBAAGjYjBgMD8
 GCWCGSAGG+EIBDQQyFjBHZW5lcmF0ZWQgYnkgdGhlIFNlY3VyaXR5IFNlcnZlciBmb3Igei9PUyA
 oUkFDRikwHQYDVR0OBBYEFLSjexfulLGxaf4xDvXV4Qhocv/JMA0GCSqGSIb3DQEBBQUAA4GBAIz
 fNvc3kWSINsVNexPANbUG9i7SR/79B++pBszHwlKsDqCcB/Sa45yIIxni6cCnLFAoKQO76wFXAnC
 Y4QDAxxukBdkiBjus0dQ4vfUDU2b5w+7F8mnvzNuHqvqBhk5DaMPbctcBl2E8lJkn3OwAk6VU+b5
 F6YJ3NT6y6SNDVk2q
objectclass: inetOrgPerson
objectclass: person
objectclass: top
```

Adding attribute types

The following example sends the Server Administration Control (OID 1.3.18.0.2.10.15) and adds two new attribute types to the existing entry. Note the registeredaddress attribute is assigned two values:

```
dn:cn=Tim Doe, ou=Your Department, o=Your Company, c=US
control: 1.3.18.0.2.10.15
changetype:modify
add:x
telephonenumber: 888 555 1234
registeredaddress: td@yourcompany.com
registeredaddress: ttd@yourcompany.com
-
```

Changing the entry name

The following example changes the name of the existing entry to cn=Tim Tom Doe, ou=Your Department, o=Your Company, c=US. The old RDN, cn=Tim Doe, is retained

as an additional attribute value of the cn attribute. The new RDN, cn=Tim Tom Doe, is added automatically by the LDAP server to the values of the cn attribute in the entry:

```
dn: cn=Tim Doe, ou=Your Department, o=Your Company, c=US
changetype:modrdn
newrdn: cn=Tim Tom Doe
deleteoldrdn: 0
```

Replacing attribute values

The following example replaces the attribute values for the telephonenumber and registeredaddress attributes with the specified attribute values.

```
dn: cn=Tim Tom Doe, ou=Your Department, o=Your Company, c=US
changetype:modify
replace:x
telephonenumber: 888 555 4321
registeredaddress: tim@yourcompany.com
registeredaddress: timtd@yourcompany.com
-
```

Deleting and adding attributes

The following example deletes the telephonenumber attribute, deletes a single registeredaddress attribute value, and adds a description attribute:

```
dn:cn=Tim Tom Doe, ou=Your Department, o=Your Company, c=US
changetype:modify
add:x
description: This is a very long attribute
  value that is continued on a second line.
   Note the spacing at the beginning of the
  continued lines in order to signify that
  the line is continued.
-
delete: telephonenumber
-
delete: registeredaddress
registeredaddress: tim@yourcompany.com
-
```

Modifying multiple entries

The following example adds the postalCode attribute and replaces the description attribute in the directory entry with name cn=Tim Tom Doe, ou=Your Department, o=Your Company, c=US and adds a new directory entry with name cn=Ken Smith, ou=Your Department, o=Your Company, c=US.

Note: A line containing only a dash is used to separate different types of changes within an entry and a blank line (a line containing no characters) is used to separate the changes to different entries.

```
dn: cn=Tim Tom Doe, ou=Your Department, o=Your Company, c=US
changetype: modify
add: x
postalcode: 12345
-
replace: x
description: This is a short description.
-

dn: cn=Ken Smith, ou=Your Department, o=Your Company, c=US
```

```
changetype: add
cn: Ken Smith
sn: Smith
objectclass: organizationalperson
```

Deleting an entry

The following example deletes the directory entry with name `cn=Tim Tom Doe, ou=Your Department, o=Your Company, c=US`:

```
dn:cn=Tim Tom Doe, ou=Your Department, o=Your Company, c=US
changetype:delete
```

Modify style

The "modify style" of input to the **ldapmodify** or **ldapadd** commands is not as flexible as the RFC 2849 LDIF style. However, it is sometimes easier to use than the LDIF style.

When using modify style input, attribute types and values are delimited by an equal sign (=). The general form of input lines for modify style is:

```
change_record
<blank line>
change_record
<blank line>
⋮
```

In modify style input:

1. A comment line is a line that begins with a number sign (#) in column 1. Comment lines are ignored.
2. A line can be continued by specifying a backslash (\) as the last character of the line. If a line is continued, the backslash character is removed and the succeeding line is appended directly after the character preceding the backslash character.

An input file in modify style consists of one or more *change_record* sets of lines separated by a single blank line. Each *change_record* has the following form:

```
distinguished_name
[+|-]{attrtype} = {value_line1[\
value_line2[\
...value_lineN]]}
⋮
```

Therefore, a *change_record* consists of a line indicating the distinguished name of the directory entry to be modified along with one or more attribute modification lines. Each attribute modification line consists of an optional add or delete indicator (+ or -), an attribute type, and an attribute value. If a plus sign (+) is specified, the modification type is set to **add**. If a hyphen (-) is specified, the modification type is set to **delete**. For a delete modification, the equal sign (=) and *value* should be omitted to remove an entire attribute. If the add or delete indicator is not specified, the modification type is set to **add** unless the **-r** option is used, in which case the modification type is set to **replace**. Any leading or trailing white space characters are removed from attribute values. If trailing white space characters are required for attribute values, the RFC 2849 LDIF style of input must be used. The new-line character at the end of the input line is not retained as part of the attribute value.

Multiple attribute values are specified by using multiple *attrtype=value* specifications.

Modify style examples

Here are some examples of valid input for the **ldapmodify** command by using modify style.

Adding a new entry

The following example adds a new entry into the directory by using name `cn=Tim Doe, ou=Your Department, o=Your Company, c=US`:

```
cn=Tim Doe, ou=Your Department, o=Your Company, c=US
cn=Tim Doe
sn=Doe
objectclass=organizationalperson
objectclass=person
objectclass=top
```

Adding a new attribute type

The following example adds two new attribute types to the existing entry. Note the `registeredaddress` attribute is assigned two values:

```
cn=Tim Doe, ou=Your Department, o=Your Company, c=US
+telephonenumber=888 555 1234
+registeredaddress=td@yourcompany.com
+registeredaddress=ttd@yourcompany.com
```

Replacing attribute values

Assuming that the command invocation was:

```
ldapmodify -r ...
```

The following example replaces the attribute values for the `telephonenumber` and `registeredaddress` attributes with the specified attribute values. If the **-r** command-line option was not specified, the attribute values are added to the existing set of attribute values.

```
cn=Tim Doe, ou=Your Department, o=Your Company, c=US
telephonenumber=888 555 4321
registeredaddress: tim@yourcompany.com
registeredaddress: timtd@yourcompany.com
```

Deleting an attribute type

The following example deletes a single `registeredaddress` attribute value from the existing entry.

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

Adding an attribute

The following example adds a `description` attribute. The `description` attribute value spans multiple lines:

```
cn=Tim Doe, ou=Your Department, o=Your Company, c=US
+description=This is a very long attribute \
value that is continued on a second line.  \
Note the backslash at the end of the line to \
be continued in order to signify that \
the line is continued.
```

Changing the numeric object identifier:

A special input file is required to change the numeric object identifier (OID) of an attribute or an object class in the z/OS LDAP server schema. This input file must contain a delete of the existing attribute or object class (with the old OID) followed by an add of the new version of the attribute or object class (with the new OID). The value for NAME within the attribute or object class must be identical in the delete and add modifications. When using the z/OS IBM Tivoli Directory Server, noncritical values (such as DESC) can be changed in the new version but critical values (such as the SYNTAX or the MUST and MAY lists) must be the same as in the existing attribute or object class. The deletion and addition must be the only modifications that are made to the schema in that operation.

For example, to change the OID for the `userHomeAddr` attribute from `1.3.21.7777` to `2.5.44.3.9999` in the schema, the input file for **ldapmodify** should contain:

```
cn=schema
-attributetypes=( 1.3.21.7777 NAME 'userHomeAddr' DESC 'The home address' \
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 USAGE userApplications )
+attributetypes=( 2.5.44.3.9999 NAME 'userHomeAddr' DESC 'The home address' \
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 USAGE userApplications )
```

Examples

Following are some **ldapmodify** and **ldapadd** examples. It might be necessary to supply a *bindDN* and *passwd* for modify to be allowed.

1. Assume that the `/tmp/entrymods` file exists and has the following contents:

   ```
   dn: cn=Modify Me, o=My Company, c=US
   changetype: modify
   replace: mail
   mail: modme@MyCompany.com
   -
   add: title
   title: Vice President
   -
   add: jpegPhoto
   jpegPhoto: /tmp/modme.jpeg
   -
   delete: description
   -
   ```

   The following command replaces the contents of the `Modify Me` entry's `mail` attribute with the value `modme@MyCompany.com`, adds a `title` of `Vice President`, adds the contents of the file `/tmp/modme.jpeg` as the `jpegPhoto` value, and completely removes the `description` attribute.

   ```
   ldapmodify -b -r -f /tmp/entrymods
   ```

   The same modifications as above can be performed by using the older **ldapmodify** input format:

   ```
   cn=Modify Me, o=My Company, c=US
   mail=modme@MyCompany.com
   +title=Vice President
   +jpegPhoto=/tmp/modme.jpeg
   -description
   ```

2. Assume that the `/tmp/certuser` file exists and has the following contents:

```
dn: cn=Karen Smith, o=My Company, c=US
objectclass: inetorgperson
cn: Karen Smith
sn: Smith
userpassword: secret
usercertificate: //'USER.CERTDER'
```

The following command adds a new entry for Karen Smith by using the values from the /tmp/certuser file. The usercertificate value is obtained from the binary data set USER.CERTDER.

```
ldapadd -b -f /tmp/certuser
```

3. Assume that the /tmp/newentry file exists and has the following contents:

```
dn: cn=Joe Smith, o=My Company, c=US
objectClass: person
cn: Joseph Smith
cn: Joe Smith
sn: Smith
title: Manager
mail: jsmith@jsmith.MyCompany.com
uid: jsmith
```

The following command adds a new entry for Joe Smith by using the values from the /tmp/newentry file.

```
ldapadd -f /tmp/newentry
```

4. Assume that the /tmp/newentry file exists and has the following contents:

```
dn: cn=Joe Smith, o=My Company, c=US
changetype: delete
```

The following command removes Joe Smith's entry.

```
ldapmodify -f /tmp/newentry
```

5. Assume that hostA contains the referral object:

```
dn: o=ABC,c=US
ref: ldap://hostB:390/o=ABC,c=US
objectclass: referral
```

and hostB contains the organization object:

```
dn: o=ABC,c=US
o: ABC
objectclass: organization
telephoneNumber: 123-4567
```

and the /tmp/refmods file has the following contents:

```
dn: o=ABC,c=US
changetype: modify
replace: ref
ref: ldap://hostB:391/o=ABC,c=US
-
```

and the /tmp/ABCmods file has the following contents:

```
dn: o=ABC,c=US
changetype: modify
add: telephoneNumber
telephoneNumber: 123-1111
-
```

The following command replaces the ref attribute value of the referral object o=ABC,c=US in hostA, changing the TCP port address in the URL from 390 to 391.

```
ldapmodify -h hostA -r -M -f /tmp/refmods
```

The following command adds the telephoneNumber attribute value 123-1111 to o=ABC,c=US in hostB.

```
ldapmodify -h hostB -p 391 -f /tmp/ABCmods
```

6. Assume that the /tmp/schemamods file exists and has the following contents:

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

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

```
ldapmodify -f /tmp/schemamods
```

7. Assume that the /tmp/newentry file exists and has the following contents:

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

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

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

8. Assume that the /tmp/modentry file contains the following attributes.

   Note: In the following LDIF, the x on the replace: x line is a placeholder for the attribute name and allows multiple attribute names and values to be replaced in a single operation.

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

   The following command adds OPERATIONS to racfattributes and changes the racfconnectgroupUACC value to update.

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

Notes

The LDAP_DEBUG environment variable can be used to set the debug level. For more information about specifying the debug level by using keywords, decimal, hexadecimal, and plus and minus syntax, see “Enabling tracing” on page 242.

You can specify an LDAP URL for *ldapHost* on the **-h** parameter. See “ldap_init()” on page 107 for more information.

For information about SSL/TLS, see “SSL/TLS information for LDAP client utilities” on page 259.

Diagnostics

Exit status is `0` if no errors occur. Errors result in a nonzero exit status and a diagnostic message being written to standard error.

ldapmodrdn utility

Purpose

The **ldapmodrdn** utility provides an interface to the **ldap_rename()** API.

The **ldapmodrdn** utility opens a connection to an LDAP server, binds, and modifies the DN of entries. The input consists of a distinguished name (DN) and a new relative distinguished name (RDN). The new RDN replaces the existing RDN in the entry that is specified by the DN. If no *dn* and *newRDN* arguments are provided, the input is read from standard input or from *file* if the **-f** option is used, and two lines are read for each rename. The first line contains the DN and the second line contains the new RDN. One or more blank lines must separate each DN and RDN pair.

The entries being renamed can be either leaf entries or non-leaf entries, and entire subtrees can be relocated in the directory with the **-s** option.

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

Format

```
ldapmodrdn [options] [dn newRDN]
```

Parameters

options

Table 12 shows the *options* you can use for the **ldapmodrdn** utility:

Table 12. ldapmodrdn options

Option	Description
-?	Print this text.
-a	Sends an `IBMModifyDNRealignDNAttributesControl` control with the operation request. The control criticality is set to `TRUE`. There is no control value. See *z/OS IBM Tivoli Directory Server Administration and Use for z/OS* for a description of this control.
-c	Continuous operation mode. Errors are reported, but **ldapmodrdn** continues with DN modifications. The return code from the utility is determined by the last DN modification. The default is to exit after reporting an error.
-d *debugLevel*	Specify the level of debug messages to be created. The debug level is specified in the same fashion as the debug level for the LDAP server. See Table 5 on page 244 for the possible values for *debugLevel*. The default is no debug messages.
-D *bindDN*	Use *bindDN* to bind to the LDAP directory. The *bindDN* parameter should be a string-represented DN. The default is a `NULL` string. If the **-S** or **-m** option is equal to `DIGEST-MD5` or `CRAM-MD5`, this option is the authorization DN that is used for making access checks. This directive is optional when used in this manner.

Table 12. ldapmodrdn options (continued)

Option	Description
-f *file*	Read the entry rename information from *file* instead of from standard input or the command-line (by specifying *dn* and *newRDN*). Multiple pairs of *dn* and *newRDN* can be specified in the input file or standard input. The pairs must be separated by one or more blank lines. Do not put quotation marks around the *dn* or *newRDN* values in the file. The *newSup* option cannot be included in *file*; this option is only accepted as a command-line option. If the *newSup* option (**-s**) is specified, each entry specified in the file has its RDN updated and be moved under the new superior entry's DN. If the `IBMModifyDNRealignDNAttributesControl` option (**-a**) is specified, it is sent on each rename operation that is specified in the file. You can specify a partitioned or sequential data set for *file* on the **-f** parameter. See "Specifying a value for a file name" on page 258 for more information.
-g *realmName*	Specify the realm name to use when doing a DIGEST-MD5 bind. This option is required when multiple realms are passed from an LDAP server to a client as part of a DIGEST-MD5 challenge; otherwise, it is optional.
-h *ldapHost*	Specify the host name or IP address on which the LDAP server is running. The default is the local host.
-k	Send the Server Administration control with the operation request. The control requires a protocol level of 3 and its criticality is set to `TRUE`. There is no control value. This control enables a server that would normally refuse updates, such as a quiesced or replica server, to allow updates. See *z/OS IBM Tivoli Directory Server Administration and Use for z/OS* for additional information about this control.
-K *keyFile*	Specify the name of the System SSL key database file, RACF key ring, or PKCS #11 token. If this option is not specified, this utility looks for the presence of the `SSL_KEYRING` environment variable with an associated name. If *keyFile* is specified as `*TOKEN*/`*`NAME`*, then System SSL uses the specified PKCS #11 token. Otherwise, System SSL uses a key database file or a RACF key ring. In this case, System SSL first assumes that *keyFile* is a key database file name and tries to locate the file. If *keyFile* is not a fully-qualified z/OS UNIX System Services file name, the current directory is assumed to contain the key database file. The name cannot be a partitioned or sequential data set. If System SSL cannot locate the file, it then assumes that *keyFile* is a RACF key ring name. See "SSL/TLS information for LDAP client utilities" on page 259 for information about System SSL key databases, RACF key rings, and PKCS #11 tokens. This parameter is ignored if **-Z** is not specified.
-l *timeLimit*	Send an `IBMModifyDNTimelimitControl` control with the operation request, substituting *timeLimit* as the control value. The control criticality is set to `TRUE`. See *z/OS IBM Tivoli Directory Server Administration and Use for z/OS* for a description of this control.

Table 12. ldapmodrdn options (continued)

Option	Description
-L	Send the Do Not Replicate control with the operation request. The control requires a protocol level of 3 and its criticality is set to `TRUE`. There is no control value. This control prevents the targeted server from sending replicated entries to the next tier of advanced replication servers. See *z/OS IBM Tivoli Directory Server Administration and Use for z/OS* for additional information about this control.
-m *mechanism*	See the description of the **-S** option.
-M	Manage referral objects as normal entries. This requires a protocol level of 3.
-n	Show what would be done, but do not actually change entries. Useful for debugging with **-v**.
-N *keyFileDN*	Specify the label associated with the certificate in the System SSL key database, RACF key ring, or PKCS #11 token. This parameter is ignored if **-Z** is not specified.
-p *ldapPort*	Specify the TCP port where the LDAP server is listening. The default LDAP non-secure port is 389 and the default LDAP secure port is 636.
-P *keyFilePW*	Specify either the key database file password or the file specification for a System SSL password stash file. When the stash file is used, it must be in the form **file://** followed immediately (no blanks) by the file system file specification (for example, `file:///etc/ldap/sslstashfile`). The stash file must be a z/OS UNIX System Services file and cannot be a partitioned or sequential data set. This parameter is ignored if **-Z** is not specified.
-r	Remove old RDN values from the entry. Default is to keep old values.
-R	Do not automatically follow referrals.
-s *newSup*	Specify the DN of the new superior entry under which the renamed entry is relocated. The *newSup* argument can be the zero-length string (**-s** `""`), if the destination server accepts zero-length string *newSup* arguments on an LDAP `Modify DN` operation.

Table 12. ldapmodrdn options (continued)

Option	Description
-S *mechanism* or **-m** *mechanism*	Specify the bind method to use. You can use either **-m** or **-S** to indicate the bind method. Specify `GSSAPI` to indicate that a Kerberos Version 5 bind is requested, `EXTERNAL` to indicate that a certificate (SASL external) bind is requested, `CRAM-MD5` to indicate that a SASL Challenge Response Authentication Mechanism bind is requested, or `DIGEST-MD5` to indicate that a SASL digest hash bind is requested. The `GSSAPI` method requires a protocol level of 3 and the user must have a valid Kerberos Ticket Granting Ticket in their credentials cache by using the Kerberos **kinit** command-line utility. The `EXTERNAL` method requires a protocol level of 3. You must also specify **-Z**, **-K**, and **-P** to use certificate bind. If there is no default certificate in the key database file, RACF key ring, or PKCS #11 token or a certificate other than the default must be used, use the **-N** option to specify the label of the certificate. The `CRAM-MD5` method requires a protocol level of 3. The **-D** or **-U** option must be specified. The `DIGEST-MD5` method requires a protocol level of 3. The **-U** option must be specified. Optionally, the **-D** option can be used to specify the authorization DN. If **-m** or **-S** is not specified, a simple bind is performed.
-U *userName*	Specify the user name for CRAM-MD5 or DIGEST-MD5 binds. The *userName* is a short name (for example, the *uid* attribute value) that is used to perform bind authentication. This option is required if the **-S** or **-m** option is set to `DIGEST-MD5`.
-v	Use verbose mode, with many diagnostics written to standard output.
-V *version*	Specify the LDAP protocol level the client should use. The value for *version* can be **2** or **3**. The default is **3**.
-w *passwd*	Use *passwd* as the password for simple, `CRAM-MD5`, and `DIGEST-MD5` authentication. The default is a `NULL` string.
-Z	Use a secure connection to communicate with the LDAP server. Secure connections expect the communication to begin with the SSL/TLS handshake. The **-K** *keyFile* option or equivalent environment variable is required when the **-Z** option is specified. The **-P** *keyFilePW* option is required when the **-Z** option is specified and the key file specifies a file system key database file. Unless you want to use the default certificate in the key database file, RACF key ring, or PKCS #11 token, use the **-N** option to specify the label of the certificate.

dn Specify the DN of the entry to change.

newRDN
Specify the new RDN for the entry.

All other command-line inputs result in a syntax error message, after which the correct syntax is displayed. If the same option is specified multiple times or if both **-m** and **-S** are specified, the last value specified is used.

Examples

The following are **ldapmodrdn** examples.

1. Assume that the /tmp/entrymods file exists and has the following contents:

   ```
   cn=Modify Me, o=My Company, c=US
   cn=The New Me
   ```

   The following command changes the RDN from cn=Modify Me to cn=The New Me and removes the old RDN cn=Modify Me from the entry. The DN of the entry is cn=The New Me, o=My Company, c=US.

   ```
   ldapmodrdn -r -f /tmp/entrymods
   ```

2. The following command is another way to accomplish the same change as Example 1. An IBMModifyDNTimelimitControl control accompanies the operation request, specifying a time limit of 30 seconds.

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

3. The following command changes the RDN from cn=Modify Me to cn=The New Me and removes the old RDN cn=Modify Me from the entry. The renamed entry is relocated beneath the new superior entry o=Some Other Company, c=US. The DN of the entry is changed to cn=The New Me, o=Some Other Company, c=US. If the renamed entry is a non-leaf node, its subordinate entries are also moved and renamed to reflect their new locations in the directory hierarchy. An IBMModifyDNTimelimitControl control accompanies the operation request, specifying a time limit of 30 seconds, and an IBMModifyDNRealignDNAttributesControl control accompanies the operation request.

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

Notes

The LDAP_DEBUG environment variable can be used to set the debug level. For more information about specifying the debug level by using keywords, decimal, hexadecimal, and plus and minus syntax, see “Enabling tracing” on page 242.

You can specify an LDAP URL for *ldapHost* on the **-h** parameter. See “ldap_init()” on page 107 for more information.

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

For information about SSL/TLS, see “SSL/TLS information for LDAP client utilities” on page 259.

Diagnostics

Exit status is 0 if no errors occur. Errors result in a nonzero exit status and a diagnostic message being written to standard error.

ldapsearch utility

Purpose

The **ldapsearch** utility provides an interface to the **ldap_search()** API.

The **ldapsearch** utility opens a connection to an LDAP server, binds, and performs a search by using the specified filter. If **ldapsearch** finds one or more entries, the specified attributes are retrieved and the entries and values are printed to standard output.

Restriction: Use of the approximate filter (~=) is not supported on a z/OS LDAP Server. This filter is processed like an equality (=) filter.

Format

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

Parameters

options

Table 13 shows the *options* you can use for the **ldapsearch** utility:

Table 13. ldapsearch options

Option	Description
-?	Print this text.
-a *deref*	Specify how alias dereferencing is done. The *deref* should be one of `never`, `always`, `search`, or `find` to specify that aliases are never dereferenced, always dereferenced, dereferenced when searching, or dereferenced only when locating the base object for the search. The default is to never dereference aliases.
-A	Retrieve attributes only (no values). This is useful when you want to see if an attribute is present in an entry and are not interested in the specific values.
-b *baseDN*	Use *baseDN* as the starting point for the search instead of the default. If **-b** is not specified, this utility examines the `LDAP_BASEDN` environment variable for a *baseDN* definition. If you are running in TSO, set the `LDAP_BASEDN` environment variable using the `_CEE_ENVFILE` Language Environment® runtime environment variable. See *z/OS XL C/C++ Programming Guide* for more information. If you are running in the z/OS shell, export the `LDAP_BASEDN` environment variable.
-B	Do not suppress display of non-printable values. This is useful when dealing with values that appear in alternate character sets such as ISO8859.1. This option is implied by the **-L** option.
-C	Do not suppress display of printable non-ASCII values (like the **-B** option). Values are displayed in the local code page. The `LANG` environment variable must be set appropriately in the shell so the characters you want print. Note the default `LANG` value of `C` causes the characters you want not to print.
-d *debugLevel*	Specify the level of debug messages to be created. The debug level is specified in the same fashion as the debug level for the LDAP server. See Table 5 on page 244 for the possible decimal values for *debugLevel*. The default is no debug messages.

Table 13. ldapsearch options (continued)

Option	Description
-D *bindDN*	Use *bindDN* to bind to the LDAP directory. The *bindDN* parameter should be a string-represented DN. The default is a `NULL` string. If the **-S** or **-m** option is equal to `DIGEST-MD5` or `CRAM-MD5`, this option is the authorization DN that is used for making access checks. This directive is optional when used in this manner.
-f *file*	Read a series of lines from *file*, performing one LDAP search for each line. In this case, the *filter* given on the command line is treated as a pattern where the first occurrence of `%s` is replaced with a line from *file*. Do not put quotation marks around the values in the file. If *file* is a single hyphen (`-`) character, then the lines are read from standard input. You can specify a partitioned or sequential data set for *file* on the **-f** parameter. See "Specifying a value for a file name" on page 258 for more information.
-F *sep*	Use *sep* as the field separator between attribute names and values. The default separator is an equal sign (=), unless the **-L** flag has been specified, in which case this option is ignored.
-g *realmName*	Specify the realm name to use when doing a DIGEST-MD5 bind. This option is required when multiple realms are passed from an LDAP server to a client as part of a DIGEST-MD5 challenge; otherwise, it is optional.
-h *ldapHost*	Specify the host name or IP address on which the LDAP server is running. The default is the local host.
-K *keyFile*	Specify the name of the System SSL key database file, RACF key ring, or PKCS #11 token. If this option is not specified, this utility looks for the presence of the `SSL_KEYRING` environment variable with an associated name. If *keyFile* is specified as `*TOKEN*/`*`NAME`*, then System SSL uses the specified PKCS #11 token. Otherwise, System SSL uses a key database file or a RACF key ring. In this case, System SSL first assumes that *keyFile* is a key database file name and tries to locate the file. If *keyFile* is not a fully-qualified z/OS UNIX System Services file name, the current directory is assumed to contain the key database file. The name cannot be a partitioned or sequential data set. If System SSL cannot locate the file, it then assumes that *keyFile* is a RACF key ring name. See "SSL/TLS information for LDAP client utilities" on page 259 for information about System SSL key databases, RACF key rings, and PKCS #11 tokens. This parameter is ignored if **-Z** is not specified.

Table 13. ldapsearch options (continued)

Option	Description
-l *timeLimit*	Limit the maximum wait time for the search request, overriding the value of `LDAP_OPT_TIMELIMIT` in the LDAP handle. Specify `NULL` for this parameter if there is no time limit for the request, or else, set the *timeLimit* value to the maximum time in seconds. The LDAP server can also provide a limit on the search time. For information about the server's search time limit and how it interacts with the client time limit, see the documentation for your LDAP server. For the z/OS LDAP servers, see the description of the **timeLimit** configuration file option (Customizing the LDAP server configuration) in *z/OS IBM Tivoli Directory Server Administration and Use for z/OS*. The default time limit for the client, which is specified by a value of `0`, indicates that there is no client time limit and that the maximum number of seconds is limited only by the LDAP server limit.
-L	Display search results in LDIF format. All characters in the LDIF format output are portable characters that are represented in the local code page. Binary attribute values are displayed in base64 encoded format. This option also turns on the **-B** option, and causes the **-F** option to be ignored. See "ldapmodify and ldapadd utilities" on page 276 for more information about the LDIF format. **Note:** LDIF output from the **ldapsearch** utility does not necessarily produce suitable data for backup and restore purposes. The **ds2ldif** utility should be considered as an alternative. The **ldapsearch** and **ds2ldif** utilities have several differences, including options to control the order of entries in the file (**ldapsearch** might not produce entries in correct hierarchical order), options to control entry contents (including operational attributes), and code pages used for portable characters in the output file. See ds2ldif utility in *z/OS IBM Tivoli Directory Server Administration and Use for z/OS* for more information.
-m *mechanism*	See the description of the **-S** option.
-M	Manage referral objects as normal entries. This requires a protocol level of 3.
-n	Show what would be done, but do not actually perform the search. Useful for debugging with **-v**.
-N *keyfileDN*	Specify the label associated with the certificate in the System SSL key database, RACF key ring, or PKCS #11 token. This parameter is ignored if **-Z** is not specified

Table 13. ldapsearch options (continued)

Option	Description
-o *sortKey*	Specifies a sort key that the client requests the server to order the results by. Multiple **-o** options can be specified to further define the sort order. The syntax of the *sortKey* parameter is as follows: `[-]attribute_name[:matching_rule_name]` where: • The optional minus sign (-) indicates to sort the results in reverse order. • *attribute_name* is the name of the attribute to sort by. • *matching_rule_name* is the optional name of a matching rule to use for sorting. The **ibm-slapdDN** attribute can be specified in a sort key to sort search results by entry DN. Matching rules are not supported by the z/OS LDAP server for a sorted search request. If specified, valid ordering rules are ignored by the z/OS LDAP server and the ordering rule associated with the attribute in the schema is used instead. This option directs the utility to send and receive the sorted search request and response controls. The criticality of the request control is always critical.
-p *ldapPort*	Specify the TCP port where the LDAP server is listening. The default LDAP non-secure port is 389 and the default LDAP secure port is 636.
-P *keyFilePW*	Specify either the key database file password or the file specification for a System SSL password stash file. When the stash file is used, it must be in the form **file://** followed immediately (no blanks) by the file system file specification (for example, `file:///etc/ldap/sslstashfile`). The stash file must be a z/OS UNIX System Services file and cannot be a partitioned or sequential data set. This parameter is ignored if **-Z** is not specified.
-q *pageSize*	Specify a page size and request that the server return search results in pages with the number of entries matching the page size. Multiple **-q** values can be specified to request pages of different sizes. In this case, the first **-q** value is used on the first page request, the second **-q** value is used on the second page request, and so on. If there are more pages than **-q** values, the last **-q** value is used for all remaining pages. The last page returned may contain fewer entries than the requested **-q** value. This option directs the utility to send and receive the paged search result control. The criticality is always critical. This option also turns on the **-R** option, which means referrals are not automatically followed.
-R	Do not automatically follow referrals.
-s *scope*	Specify the scope of the search. The *scope* should be one of `base`, `one`, or `sub` to specify a base object, one-level, or subtree search. The default is `sub`.

Table 13. ldapsearch options (continued)

Option	Description
-S *mechanism* or **-m** *mechanism*	Specify the bind method to use. You can use either **-m** or **-S** to indicate the bind method. Specify `GSSAPI` to indicate that a Kerberos Version 5 bind is requested, `EXTERNAL` to indicate that a certificate (SASL external) bind is requested, `CRAM-MD5` to indicate that a SASL Challenge Response Authentication Mechanism bind is requested, or `DIGEST-MD5` to indicate that a SASL digest hash bind is requested. The `GSSAPI` method requires a protocol level of 3 and the user must have a valid Kerberos Ticket Granting Ticket in their credentials cache by using the Kerberos **kinit** command line utility. The `EXTERNAL` method requires a protocol level of 3. You must also specify **-Z**, **-K**, and **-P** to use certificate bind. If there is no default certificate in the key database file, RACF key ring, or PKCS #11 token or a certificate other than the default must be used, use the **-N** option to specify the label of the certificate. The `CRAM-MD5` method requires a protocol level of 3. The **-D** or **-U** option must be specified. The `DIGEST-MD5` method requires a protocol level of 3. The **-U** option must be specified. Optionally, the **-D** option can be used to specify the authorization DN. If **-m** or **-S** is not specified, a simple bind is performed.
-t	Write retrieved values to a set of files in the `/tmp` directory, using file names like `/tmp/ldapsearch-objectclass-bbeFxQ`. This option assumes that values are non-textual (binary), such as `jpegPhoto` or `audio`. There is no character set translation performed on the values.
-T *pageTime*	Specify the number of seconds between paged search requests. This option requires the **-q** option. An alternative to the **-T** option for requesting a subsequent page is to press the Enter key after paged results are returned. If the utility is being executed from a batch job and the **-q** option is specified, the **-T** option must be used.
-U *userName*	Specify the user name for CRAM-MD5 or DIGEST-MD5 binds. The *userName* is a short name (for example, the *uid* attribute value) that is used to perform bind authentication. This option is required if the **-S** or **-m** option is set to `DIGEST-MD5`.
-v	Run in verbose mode, with many diagnostics written to standard output.
-V *version*	Specify the LDAP protocol level the client should use. The value for *version* can be **2** or **3**. The default is **3**.
-w *passwd*	Use *passwd* as the password for simple, CRAM-MD5, and DIGEST-MD5 authentication. The default is a `NULL` string.

Table 13. ldapsearch options (continued)

Option	Description
-z *sizeLimit*	Limit the number of entries that can be returned, overriding the value of `LDAP_OPT_SIZELIMIT` in the LDAP handle. A value of `0` indicates that there is no limit. The LDAP server can also provide a size limit on the number of entries returned. For information about the server's size limit and how it interacts with the client size limit, see the documentation for your LDAP server. For the z/OS LDAP servers, see the description of the **sizeLimit** configuration file option (Customizing the LDAP server configuration) in *z/OS IBM Tivoli Directory Server Administration and Use for z/OS*. The default size limit for the client, specified by a value of `0`, indicates that the maximum number of entries is limited only by the LDAP server limit.
-Z	Use a secure connection to communicate with the LDAP server. Secure connections expect the communication to begin with the SSL/TLS handshake. The **-K** *keyFile* option or equivalent environment variable is required when the **-Z** option is specified. The **-P** *keyFilePW* option is required when the **-Z** option is specified and the key file specifies a file system key database file. Unless you want to use the default certificate in the key database file, RACF key ring, or PKCS #11 token, use the **-N** option to specify the label of the certificate.

filter
: Specify an IETF RFC 1558-compliant LDAP search filter. (See "ldap_search(), ldap_search_s(), ldap_search_st(), ldap_search_ext(), ldap_search_ext_s()" on page 164 for more information about filters.)

attributes
: Specify a space-separated list of attributes to retrieve. If no *attributes* list is given, all are retrieved.

All other command line inputs result in a syntax error message, after which the correct syntax is displayed. If the same option is specified multiple times or if both **-m** and **-S** are specified, the last value specified is used.

Output format

If one or more entries are found, each entry is written to standard output in the form:

```
Distinguished Name (DN)
attributename=value
attributename=value
attributename=value
...
```

Multiple entries are separated with a single blank line. If the **-F** option is used to specify a separator character, it is used instead of the equal sign (=). If the **-t** option is used, the name of a temporary file is used in place of the actual value. If the **-A** option is given, only the `attributename` part is written.

Examples

Following are some **ldapsearch** examples. Each example makes the assumption that the LDAP server is running on the local host and listening on the default LDAP port (389).

- The command:

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

  performs a subtree search using the search base "o=IBM University,c=US" for entries with a commonName of karen smith. The commonName and telephoneNumber values are retrieved and printed to standard output. The output might look something like this if two entries are found:

  ```
  cn=Karen G Smith, ou=College of Engineering, o=IBM University, c=US
  cn=Karen Smith
  cn=Karen Grace Smith
  cn=Karen G Smith
  telephoneNumber=+1 313 555-9489

  cn=Karen D Smith, ou=Information Technology Division, o=IBM University, c=US
  cn=Karen Smith
  cn=Karen Diane Smith
  cn=Karen D Smith
  telephoneNumber=+1 313 555-2277
  ```

- The command:

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

  Performs a subtree search using the search base "o=IBM University,c=US" for entries with user ID of kds. The jpegPhoto and audio values are retrieved and written to temporary files. The output might look like this if one entry with one value for each of the requested attributes is found:

  ```
  cn=Karen D Smith, ou=Information Technology Division, o=IBM University, c=US
  audio=/tmp/ldapsearch-audio-a19924
  jpegPhoto=/tmp/ldapsearch-jpegPhoto-a19924
  ```

- The command:

  ```
  ldapsearch -L -b "c=US" "(&(usercertificate=*)(objectclass=inetOrgPerson))" usercertificate cn
  ```

  Performs a subtree-level search at the c=US level for all users that have an objectclass value of inetOrgPerson and a userCertificate value. Search results are displayed in the LDIF format. The userCertificate attribute value is displayed in base64-encoded format because it is a binary value. The userCertificate and cn attribute values are retrieved and printed to standard output, resulting in output like the following:

  ```
  dn: cn=John Doe, ou=Your Department, o=Your Company, c=US
  cn: John Doe
  usercertificate:: MIICNjCCAZ+gAwIBAgIBADANBgkqhkiG9w0BAQUFADAvMQswCQYDVQQGEwJ
   1czEMMAoGA1UEChMDaWJtMRIwEAYDVQQDEwlyMTNzZXJ2ZXIwHhcNMTAwNjE1MDQwMDAwWhcNMjE
   wMTAxMDM1OTU5WjAvMQswCQYDVQQGEwJ1czEMMAoGA1UEChMDaWJtMRIwEAYDVQQDEwlyMTNzZXJ
   2ZXIwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMVgV8f3IAZEZ5/h3R2Iy7h4LSHbhsj4diH
   iHPIpTRtqJD5d42z2Z4gG9oUzqfYLyZSPoAVlDwVbufZVVvBeiDo7Bgm+1nj4/YYWCpnCkETmriB
   bVDJBoaF8W9xxHs38F6LVuJniDMp0VT9lDcqH3RNWgIcDqKurQm2uTHNDs6OtAgMBAAGjYjBgMD8
   GCWCGSAGG+EIBDQQyFjBHZW5lcmF0ZWQgYnkgdGhlIFNlY3VyaXR5IFNlcnZlciBmb3Igei9PUyA
   oUkFDRikwHQYDVR0OBBYEFLSjexfulLGxaf4xDvXV4Qhocv/JMA0GCSqGSIb3DQEBBQUAA4GBAIz
   fNvc3kWSINsVNexPANbUG9i7SR/79B++pBszHwlKsDqCcB/Sa45yIIxni6cCnLFAoKQO76wFXAnC
   Y4QDAxxukBdkiBjus0dQ4vfUDU2b5w+7F8mnvzNuHqvqBhk5DaMPbctcBl2E8lJkn3OwAk6VU+b5
   F6YJ3NT6y6SNDVk2q
  ```

- The command:

  ```
  ldapsearch -L -s one -b "c=US" "o=university*" o description
  ```

  Performs a one-level search at the c=US level for all organizations whose organizationName begins with university. Search results are displayed in the LDIF format. The organizationName and description attribute values are retrieved and printed to standard output, resulting in output like the following:

```
dn: o=University of Alaska Fairbanks, c=US
o: University of Alaska Fairbanks
description: Preparing Alaska for a brave new tomorrow
description: leaf node only

dn: o=University of Colorado at Boulder, c=US
o: University of Colorado at Boulder
description: No personnel information
description: Institution of education and research

dn: o=University of Colorado at Denver, c=US
o: University of Colorado at Denver
o: UCD
o: CU/Denver
o: CU-Denver
description: Institute for Higher Learning and Research

dn: o=University of Florida, c=US
o: University of Florida
o: UFl
description: Shaper of young minds
...
```

- The command:

  ```
  ldapsearch -h ushost -M -b "c=US"  "objectclass=referral"
  ```

  Performs a subtree search for the c=US subtree within the server at host ushost (TCP port 389) and returns all referral objects. Note that the search is limited to the single server. No referrals are followed to other servers to find additional referral objects. The output might look something like this if two referral objects are found:

  ```
  o=IBM,c=US
  objectclass=referral
  ref=ldap://ibmhost:389/o=IBM,c=US

  o=XYZ Company,c=US
  objectclass=referral
  ref=ldap://XYZhost:390/o=XYZ%20Company,c=US
  ```

- The command:

  ```
  ldapsearch -b "o=Deltawing, c=AU" -o sn -o -ibm-slapdDN -q 3 -q 2 -T 10
   -v "(|(sn=Harris)(sn=Stephens))" sn
  ```

  Performs a verbose sorted and paged search sorting first by surname, then by distinguished name, with the distinguished name being sorted in reverse (descending) order as specified by the prefixed minus sign. The first page contains 3 entries. The second page contains 2 entries. The last page contains the final entry, where six entries were found. Each subsequent page is requested 10 seconds after the preceding page is returned. The output might look something like this:

  Note: In the following example output, text in this **format** is the returned search entries, while text in this format is the verbose **ldapsearch** utility output.

  ```
  -q option implies -R option. Referrals will not be followed.
  ldap_init(MYHOST, 389)
  filter pattern: (|(sn=Harris)(sn=Stephens))
  returning: sn
  sorted search keys: sn -ibm-slapdDN
  paged search sizes: 3 2
  paged search time: 10

  filter is: ((|(sn=Harris)(sn=Stephens)))
  cn=Virginia Harris, o=Deltawing, c=AU
  sn=Harris
  ```

```
cn=Michael Harris, o=Deltawing, c=AU
sn=Harris

cn=Martin Harris, o=Deltawing, c=AU
sn=Harris
3 matches
3 total paged entries have been returned
6 entries in entire paged results set
The next page will be retrieved in 10 seconds...

cn=Paul Stephens, o=Deltawing, c=AU
sn=Stephens

cn=David Stephens, o=Deltawing, c=AU
sn=Stephens
2 matches
5 total paged entries have been returned
6 entries in entire paged results set
The next page will be retrieved in 10 seconds...

cn=Brian Stephens, o=Deltawing, c=AU
sn=Stephens
1 matches
6 total paged entries have been returned
6 entries in entire paged results set
```

- The command:

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

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

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

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

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

  The results might look like:

```
racfid=GLDSRV,profiletype=USER,sysplex=sysplexa
racfid=GLDSRV
racfauthorizationdate=05/15/07
racfowner=RACFID=SUIMGVD,PROFILETYPE=USER,SYSPLEX=SYSPLEXA
racfpasswordinterval=186
racfdefaultgroup=RACFID=AUDIT,PROFILETYPE=GROUP,SYSPLEX=SYSPLEXA
racflogondays=SUNDAY
racflogondays=MONDAY
racflogondays=TUESDAY
racflogondays=WEDNESDAY
racflogondays=THURSDAY
racflogondays=FRIDAY
racflogondays=SATURDAY
racflogontime=ANYTIME
racfconnectgroupname=RACFID=AUDIT,PROFILETYPE=GROUP,SYSPLEX=SYSPLEXA
racfhavepasswordenvelope=NO
racfhavepassphraseenvelope=NO
```

```
racfattributes=PASSWORD
objectclass=TOP
objectclass=RACFBASECOMMON
objectclass=RACFUSER
```

The following examples use file input to perform multiple searches with the same search base and scope, but with different filters. Each line in the file is used to replace the first occurrence of %s in the filter. The %s can be anywhere in the filter, and can be the entire filter.

- Assume file /tmp/searchFile has the following contents:

  ```
  Smith
  Jones
  Doe
  ```

  The command:

  ```
  ldapsearch -f /tmp/searchFile -L -s sub -b "o=My Company" "(&(cn=John)(sn=%s*))"
  ```

  replaces the %s in the filter value with each line of the input file. This is equivalent to issuing these search commands:

  ```
  ldapsearch -L -s sub -b "o=My Company" "(&(cn=John)(sn=Smith*))"
  ldapsearch -L -s sub -b "o=My Company" "(&(cn=John)(sn=Jones*))"
  ldapsearch -L -s sub -b "o=My Company" "(&(cn=John)(sn=Doe*))"
  ```

- Assume file /tmp/searchFile has the following contents:

  ```
  o=university*
  cn=Karen Smith
  ```

  The command:

  ```
  ldapsearch -f /tmp/searchFile -s one -b "c=US" "%s" description
  ```

  replaces the entire filter with each line of the input file. This is equivalent to issuing these search commands:

  ```
  ldapsearch -s one -b "c=US" "o=university*" description
  ldapsearch -s one -b "c=US" "cn=Karen Smith" description
  ```

Searching a server's root DSE

The command:

```
ldapsearch -h ushost -s base -b ""  "objectclass=*"
```

provides the root DSE (DSA-specific entry, where a DSA is a directory server) information for a server. This request can be directed to servers supporting LDAP Version 3 protocol to obtain information about support available in the server. See IETF RFC 2251: *Lightweight Directory Access Protocol (v3)* for a description of the information provided by the server. See *z/OS IBM Tivoli Directory Server Administration and Use for z/OS* for more information about the root DSE and what the IBM Tivoli Directory Server for z/OS returns.

The command:

```
ldapsearch -h ushost -s sub -b "" filter
```

searches the directories within the LDAP server for entries that match the filter. This type of search is commonly referred to as a null-based subtree search. See *z/OS IBM Tivoli Directory Server Administration and Use for z/OS* for more information about the z/OS LDAP server support for null-based subtree searches.

Note: The scope option **(-s)** must be specified when specifying `-b ""` to search a server's root DSE.

Notes

The `LDAP_DEBUG` environment variable can be used to set the debug level. For more information about specifying the debug level using keywords, decimal, hexadecimal, and plus and minus syntax, see “Enabling tracing” on page 242.

You can specify an LDAP URL for *ldapHost* on the **-h** parameter. See “ldap_init()” on page 107 for more information.

For information about SSL/TLS, see “SSL/TLS information for LDAP client utilities” on page 259.

Diagnostics

Exit status is `0` if no errors occur. Errors result in a nonzero exit status and a diagnostic message being written to standard error.

Appendix. Accessibility

Accessible publications for this product are offered through IBM Knowledge Center (http://www.ibm.com/support/knowledgecenter/SSLTBW/welcome).

If you experience difficulty with the accessibility of any z/OS information, send a detailed message to the "Contact us" web page for z/OS (http://www.ibm.com/systems/z/os/zos/webqs.html) or use the following mailing address.

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
United States

Accessibility features

Accessibility features help users who have physical disabilities such as restricted mobility or limited vision use software products successfully. The accessibility features in z/OS can help users do the following tasks:

- Run assistive technology such as screen readers and screen magnifier software.
- Operate specific or equivalent features by using the keyboard.
- Customize display attributes such as color, contrast, and font size.

Consult assistive technologies

Assistive technology products such as screen readers function with the user interfaces found in z/OS. Consult the product information for the specific assistive technology product that is used to access z/OS interfaces.

Keyboard navigation of the user interface

You can access z/OS user interfaces with TSO/E or ISPF. The following information describes how to use TSO/E and ISPF, including the use of keyboard shortcuts and function keys (PF keys). Each guide includes the default settings for the PF keys.

- *z/OS TSO/E Primer*
- *z/OS TSO/E User's Guide*
- *z/OS V2R2 ISPF User's Guide Vol I*

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users who access IBM Knowledge Center with a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line because they are considered a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that the screen reader is set to read out

punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1) are mutually exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The * symbol is placed next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is given the format 3 * FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* * FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol to provide information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, it indicates a reference that is defined elsewhere. The string that follows the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you must refer to separate syntax fragment OP1.

The following symbols are used next to the dotted decimal numbers.

? indicates an optional syntax element
The question mark (?) symbol indicates an optional syntax element. A dotted decimal number followed by the question mark symbol (?) indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that the syntax elements NOTIFY and UPDATE are optional. That is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

! indicates a default syntax element
The exclamation mark (!) symbol indicates a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicate that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the dotted decimal number can specify the ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the

default option for the FILE keyword. In the example, if you include the FILE keyword, but do not specify an option, the default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, the default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1.1 (DELETE), the default option KEEP applies only to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

*** indicates an optional syntax element that is repeatable**

The asterisk or glyph (*) symbol indicates a syntax element that can be repeated zero or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3* , 3 HOST, 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

Notes:

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you can write HOST STATE, but you cannot write HOST HOST.
3. The * symbol is equivalent to a loopback line in a railroad syntax diagram.

+ indicates a syntax element that must be included

The plus (+) symbol indicates a syntax element that must be included at least once. A dotted decimal number followed by the + symbol indicates that the syntax element must be included one or more times. That is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the * symbol, the + symbol can repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loopback line in a railroad syntax diagram.

Notices

This information was developed for products and services offered in the U.S.A. or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, 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:

Site Counsel
IBM Corporation
2455 South Road
Poughkeepsie, NY 12601-5400
USA

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.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

COPYRIGHT LICENSE:

This information might contain sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, HCD, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted

for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (http://www.ibm.com/software/support/systemsz/lifecycle/)
- For information about currently-supported IBM hardware, contact your IBM representative.

Programming interface information

z/OS IBM Tivoli Directory Server Client Programming for z/OS primarily documents intended Programming Interfaces that allow the customer to write programs to obtain services of z/OS LDAP.

z/OS IBM Tivoli Directory Server Plug-in Reference for z/OS also documents information that is not intended to be used as Programming Interfaces of z/OS LDAP. This information is identified where it occurs with an introductory statement to a chapter.

Programming interface information

End Programming interface information

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available at Copyright and Trademark information (http://www.ibm.com/legal/copytrade.shtml).

Index

Special characters

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Product Number: 5650-ZOS

Printed in USA

SA23-2295-01