IBM Directory Server Version 4.1: C-Client SDK Programming Reference

IBM Directory Server Version 4.1: C-Client SDK Programming Reference

Note

Before using this information and the product it supports, read the general information under Appendix E, "Notices" on page 191.

First Edition (April 2002)

This edition applies to version 4, release 1, of the IBM Directory Server and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Preface

The IBM® Directory Server C-Client SDK includes various sample LDAP client programs, and an LDAP client library used to provide application access to the LDAP servers.

Chapter 1. IBM Directory C-Client SDK overview

The Lightweight Directory Access Protocol (LDAP) provides TCP/IP access to LDAP-compliant servers. The IBM Directory Server C-Client SDK includes various sample LDAP client programs, and an LDAP client library used to provide application access to the LDAP servers.

See the following for more information:

- LDAP Version Support
- LDAP API Overview
- Updates for IBM Directory Server C-Client Version 4.1

LDAP version support

The IBM Directory Server C-Client SDK provides support for both LDAP Version 2 and LDAP Version 3 application programming interfaces (APIs) and protocols. The LDAP SDK APIs are based upon the Internet Draft, ″C LDAP Application Program Interface ″, which is classified as a work in progress.

The LDAP API provides typical directory functions such as read, write and search. With the advent of support for LDAP Version 3 APIs and protocols, the following features are also supported:

- LDAP V3 referrals and search references.
- Improved internationalization with UTF-8 support for Distinguished Names (DNs) and strings that are passed into, and returned from, the LDAP APIs. Support for converting string data between the local code page and UTF-8 is also provided. When running as an LDAP V2 application, DNs and strings remain limited to the IA5 character set.
- As provided by the IBM Directory server's dynamic schema capability, an LDAP application can add, modify and change elements of the schema (see Appendix A, "LDAP V3 schema" on page 177) for more information).
- Controls for the LDAP server and client

With the C-Client SDK, an application that uses the ldap_open API defaults to the LDAP V2 protocol. Existing LDAP applications continue to work, and can interoperate with both LDAP V2 servers and LDAP V3 servers.

An application that uses the ldap_init API defaults to the LDAP V3 protocol with optional bind. An LDAP V3 application does not necessarily interoperate with an LDAP server that supports only LDAP V2 protocols.

Note: An application can use the ldap_set_option API to change its LDAP protocol version. This is done after using ldap_open or ldap_init but before issuing a bind or any other operation that results in contacting the server.

LDAP API overview

The set of LDAP APIs is designed to provide a suite of functions that can be used to develop directory-enabled applications. Directory-enabled applications are typically connect to one or more directories and perform various directory-related operations, such as:

- Adding entries
- Searching the directories and obtaining the resulting list of entries
- Deleting entries
- Modifying entries
- Renaming entries

The type of information that is managed in the directory depends on the nature of the application. Directories often are used to provide public access to information about people. For example:

- phone numbers
- e-mail addresses
- fax numbers
- mailing addresses

Increasingly, directories are being used to manage and publish other types of information. For example:

- Configuration information
- Public key certificates (managed by certification authorities (CAs))
- Access control information
- Locating information (how to find a service)

The LDAP API provides for both synchronous and asynchronous access to a directory. Asynchronous access enables your application to do other work while waiting for the results of a directory operation to be returned by the server.

The source code, example makefile and executable programs are provided that perform the following operations:

- **ldapsearch** (searches the directory)
- **ldapmodify** (modifies information in the directory)
- **ldapdelete** (deletes information from the directory)
- **ldapmodrdn** (modifies the Relative Distinguished Name (RDN) of an entry in the directory)

Typical API usage

The basic interaction is as follows:

1. A connection is made to an LDAP server by calling ldap_init (or ldap_ssl_init which is used to establish a secure connection over Secure Sockets Layer (SSL).
2. An LDAP bind operation is performed by calling ldap_simple_bind. The bind operation is used to authenticate to the directory server. Note that the LDAP V3 API and protocol permits the bind to be skipped, in which case the access rights associated with anonymous access are obtained.
3. Other operations are performed by calling one of the synchronous or asynchronous routines (for example, ldap_search_s or ldap_search followed by ldap_result).
4. Results returned from these routines are interpreted by calling the LDAP parsing routines, which include operations such as:
   - ldap_first_entry, ldap_next_entry
   - ldap_get_dn
   - ldap_first_attribute, ldap_next_attribute

- ldap_get_values
- ldap_parse_result (new for LDAP V3)

5. The LDAP connection is terminated by calling ldap_unbind.

When handling a client referral to another server, the ldap_set_rebind_proc routine defines the entry-point of a routine called when an LDAP bind operation is needed.

Displaying results

Results obtained from the ldap search routines can be accessed by calling:

- ldap_first_entry and ldap_next_entry to step through the entries returned
- ldap_first_attribute and ldap_next_attribute to step through an entry's attributes
- ldap_get_values to retrieve a given attribute's value
- printf or some other display or usage method

Uniform Resource Locators (URLs)

Use the ldap_url routines to test a URL to see if it is an LDAP URL, to parse LDAP URLs into their component pieces, and to initiate searches directly using an LDAP URL. Some examples of these routines are ldap_url_parse, ldap_url_search_s, and ldap_is_ldap_url.

Secure Socket Layer (SSL) support

Note: This function is not supported on the Linux platform.

The LDAP API has been extended to support connections that are protected by the SSL protocol. This can be used to provide strong authentication between the client and server, as well as data encryption of LDAP messages that flow between the client and the LDAP server. The ldap_ssl_client_init() and ldap_ssl_init() APIs are provided to initialize the SSL function, and to create a secure SSL connection.

Updates for IBM Directory Server C-Client Version 4.1

The following are enhancements available with the IBM Directory Server C-Client Version 4.1.

Client DN processing functions

The client DN processing functions normalize attribute values that contain compound RDNs, escaped hex representations of UTF-8 characters and ber-encoded values. The functions also check that the DN passed in is in a correct format according to RFC 2253. ldap_explode_rdn removes back slashes (\ ) from in front of special characters.

ldap_dn2ufn, ldap_explode_dn and ldap_explode_rdn normalize attribute values by doing the following:

- A back slash followed by a two-digit hex representation of a UTF-8 character is converted to the character representation. For example, `cn=\4A\6F\68\6E Doe` is converted to `cn=John Doe`.
- A ber-encoded value is converted to a UTF-8 value. For example, `cn=#04044A6F686E20446F65` is converted to `cn=John Doe`.

ldap_dn2ufn, ldap_explode_dn and ldap_explode_rdn check that the DN passed in is valid. If the DN is invalid, NULL is returned. A DN is invalid if the attribute type or value are in invalid formats. See RFC 2253 for more specific information.

ldap_dn2ufn, ldap_explode_dn and ldap_explode_rdn now handle compound RDNs correctly. For example:

- The DN `cn=John+sn=Doe` passed into ldap_dn2ufn returns `John+Doe`
- ldap_explode_dn with notype returns `John+Doe`
- ldap_explode_rdn with notype returns `[0]=John [1]=Doe`

ldap_explode_rdn removes the back slash from in front of special characters. For example, when calling ldap_explode_rdn(cn=Doe\<Jane+ou=LDAP+o=IBM+c=US,1), ldap_explode_rdn returned:

- [0] = Doe<Jane
- [1] = LDAP
- [2] = IBM
- [3] = US

Kerberos 1.2

For IBM Directory Server Version 4.1, Kerberos 1.2 is used on the AIX® operating systems. For IBM Directory Server Version 4.1, Kerberos 1.1 is used on the Windows NT® and Windows® 2000 operating systems. IBM Directory Server version 4.1 does not support Kerberos authentication on the Solaris or HP operating systems.

SSL

For IBM Directory Server C-Client Version 4.1, the 56-bit version of SSL is removed. SSL is only available on a 128-bit cipher.

Sorted Search and Paged Results

There are several new APIs that can be used by client applications to request sorted search results or simple paged results of search entries. Both of these functions are requested by the client application through the use of LDAP controls specified when the search request is submitted to the server.

Server side sorting of search results

Sorted Search Results provides sort capabilities for LDAP clients that have limited or no sort functionality. Sorted Search Results enables an LDAP client to receive sorted search results based on a list of criteria, where each criteria represents a sort key. The sort criteria includes attribute types, matching rules, or descending order. The server must use this criteria to sort search results before returning them. This moves the responsibility of sorting from the client application to the server, where it might be done much more efficiently. For example, a client application might want to sort the list of employees at their Grand Cayman site by surname, common name, and telephone number. Instead of building the search list twice so it can be sorted (once at the server and then again at the client when all the results are returned), the search list is built once, and then sorted, before returning the results to the client application.

There are four new APIs that can be used by a client application to request sorted search results:

- ldap_create_sort_key_list()

- ldap_create_sort_control()
- ldap_free_sort_keylist()
- ldap_parse_sort_control()

Details about these APIs can be found in "LDAP_SORT" on page 123. The ldap_create_sort_key_list() API builds a list of LDAPsortkey structures based on the list of attributes included in the incoming string. A sort key is made up of three possible values:

- Name of attribute used to sort entries returned by the server
- Object identifier (OID) of a matching rule for that attribute
- Whether or not the sort must be done in reverse order

The syntax of the sortString used as input to the ldap_create_sort_key_list() API is:

```
[-]<attribute name>[:<matching rule OID>]
```

where `<attribute name>` is the attribute used to perform the sort, `<matching rule OID>` is the OID to be used when sorting, and the optional prefixed minus sign (-) indicates that the sort must be done in reverse order. Only the attribute name is required. In the following example sortString, the search results are sorted first by surname (sn), then by given name (givenname), with the given name being sorted in reverse (descending) order as specified by the prefixed minus sign (-).

```
sn -givenname
```

The sortKeyList output from ldap_create_sort_key_list() can be used as input to ldap_create_sort_control(). The sortKeyList is an ordered array of LDAPsortkey structures such that the key with the highest precedence is at the front of the array. ldap_create_sort_control() outputs a LDAPControl structure which can be added to the list of client controls sent to the server on the LDAP search request. The LDAPControl structure returned by the ldap_create_sort_control() API can be used as input to ldap_search_ext() or ldap_search_ext_s(), which are used to make the actual search request.

Note: Server side sorting is an optional extension of the LDAP v3 protocol, so the server you have bound to prior to the ldap_search_ext() or ldap_search_ext_s() call might not support this function.

Now that you have created the server side control, you can free the sortKeyList output from ldap_create_sort_key_list() using ldap_free_sort_keylist().

Upon completion of the search request you submitted using the ldap_search_ext() or ldap_search_ext_s(), the server returns an LDAP result message that includes a sort results control. The client application can parse this control using ldap_parse_sort_control() which takes the returned server response controls (a null terminated array of pointers to LDAPControl structures) as input. ldap_parse_sort_control() outputs a return code which indicates whether or not the sort request was successful. If the sort was not successful, the name of the attribute in error might be output from ldap_parse_sort_control(). Use ldap_controls_free() to free the memory used by the client application to hold the server controls when you are done processing all controls returned by the server for this search request.

The server returns a successful return code of LDAP_SUCCESS in the sort response control (sortKeyResponseControl) in the search result (searchResultDone) message if the server supports sorting and can sort the search results using the specified keys. If the search fails for any reason or there are no search results, then the server omits the sortKeyResponseControl from the searchResultsDone message.

If the server does not support sorting and the criticality specified on the sort control for the search request is TRUE, the server does not return any search results, and the sort response control return code is set to LDAP_UNAVAILABLE_CRITICAL_EXTENSION. If the server does not support sorting and the criticality specified on the sort control for the search request is FALSE, the server returns all search results and the sort control is ignored.

If the server does support sorting and the criticality specified on the sort control for the search request is TRUE, but for some reason cannot sort the search results, then the sort response control return code is set to LDAP_UNAVAILABLE_CRITICAL_EXTENSION and no search results are returned. If the server does support sorting and the criticality specified on the sort control for the search request is FALSE, and for some reason cannot sort the search results, then the sort response control return code is set to the appropriate return code and all search results are returned unsorted.

The following return codes might be returned by the server in the sortKeyResponseControl of the searchResultDone message:

- LDAP_SUCCESS - the results are sorted
- LDAP_OPERATIONS_ERROR - server internal failure
- LDAP_TIMELIMIT_EXCEEDED - time limit reached before sorting was completed
- LDAP_STRONG_AUTH_REQUIRED - refused to return sorted results using insecure protocol
- LDAP_ADMIN_LIMIT_EXCEEDED - too many matching entries for the server to sort
- LDAP_NO_SUCH_ATTRIBUTE - unrecognized attribute type in sort key
- LDAP_INAPPROPRIATE_MATCHING - unrecognized or inappropriate matching rule in sort key
- LDAP_INSUFFICIENT_ACCESS - refused to return sorted results to this client
- LDAP_BUSY - too busy to process
- LDAP_UNWILLING_TO_PERFORM - unable to sort
- LDAP_OTHER - unable to sort due to reasons other than those specified above

There are other rules that must be taken into consideration when requesting sort from the server, they include the following:

- The matching rule must be one that is valid for the sort attribute it applies to. The server returns LDAP_INAPPROPRIATE_MATCHING if it is not.
- If the matching rule is omitted from a sort key, the ordering matching rule defined for use with this sort attribute must be used.
- A server can restrict the number of keys supported for a sort control, such as supporting only one key (a sort key list of at least one key must be supported).
- If a search result meets the search criteria but is missing a value for the sort key (sort attribute value is NULL), then this search result is considered a larger value than any other valid values for that key.

When sorted search is requested along with simple paged results, the sortKeyResponseControl is returned on every searchResultsDone message, not just the last one of the paged results request. Of course the sortKeyResponseControl might not be returned if there is an error processing the paged results request or there are no search results to return. Additionally, when sorted search is requested along with simple paged results, the server sends the search results sorted based on the entire search result set and not just simply sort each page.

When chasing referrals, the client application needs to send in a sorted search request to each of the referral servers. It is up to the application using the client's services to decide whether or not to set the criticality as to the support of sorted search results, and to handle a lack of support of this control on referral servers as appropriate based on the application. Additionally, the LDAP server does not ensure that the referral server supports the sorted search control. Multiple lists might be returned to the client application, some of which are not sorted. It is the client application's decision as to how best to present this information to the end user. Possible solutions include:

- Combine all referral results before presenting to the end user
- Show multiple lists and the corresponding referral server host name
- Take no extra steps and show all results to the end user as they are returned from the server

The client application must turn off referrals to get one truly sorted list, otherwise when chasing referrals with the sorted search control specified, unpredictable results can occur.

More information about the server side sorted search control, with control OID of 1.2.840.113556.1.4.473, can be found in RFC 2891 - LDAP Control Extension for Server Side Sorting of Search Results.

Simple paged results of search results

Simple Paged Results 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. The server ignores a simple paged results request if the page size is greater than or equal to the sizeLimit value for the server because the request can be satisfied in a single operation.

There are two new APIs that can be used by a client application to request paging of search results:

- ldap_create_page_control()
- ldap_parse_page_control()

Details about these APIs can be found in "LDAP_PAGED_RESULTS" on page 126. The ldap_create_page_control() API takes as input a page size and a cookie, and outputs an LDAPControl structure which can be added to the list of client controls sent to the server on the LDAP search request. The page size specifies how many search results must be returned for this request, and the cookie is an opaque structure returned by the server (on the initial paged results search request, the cookie must be a zero-length string). No assumptions must be made about the internal organization or value of the cookie. The cookie is used on subsequent paged results search requests when more entries are to be retrieved from the results set. The cookie must be the value of the cookie returned on the last response returned from the server on all subsequent paged results search requests. The cookie is empty when there are no more entries to be returned by the server, or when the client application abandons the paged results request by sending in a zero page size. Once the paged results search request has been completed, the cookie must not be used because it is no longer valid.

The LDAPControl structure returned by ldap_create_page_control() can be used as input to ldap_search_ext() or ldap_search_ext_s(), which are used to make the actual search request.

Note: Server side simple paged results is an optional extension of the LDAP v3 protocol, so the server you have bound to prior to the ldap_search_ext() or ldap_search_ext_s() call might not support this function.

Upon completion of the search request you submitted using ldap_search_ext() or ldap_search_ext_s(), the server returns an LDAP result message that includes a paged results control. The client application can parse this control using ldap_parse_page_control() which takes the returned server response controls (a null terminated array of pointers to LDAPControl structures) as input. ldap_parse_page_control() outputs a cookie and the total number of entries in the entire search result set. Servers that cannot provide an estimate for the total number of entries might set this value to zero (0). Use ldap_controls_free() to free the memory used by the client application to hold the server controls when you are done processing all controls returned by the server for this search request.

The server might limit the number of outstanding paged results operations from a given client or for all clients. A server with a limit on the number of outstanding paged results requests might return either LDAP_UNWILLING_TO_PERFORM in the sortResultsDone message or age out an older paged results request. There is no guarantee to the client application that the results of a search query have remained unchanged throughout the life of a set of paged results request/response sequences. If the result set for that query has changed since the initial search request specifying paged results, the client application might not receive all the entries matching the given search criteria. When chasing referrals, the client application needs to send in an initial paged results request, with the cookie set to null, to each of the referral servers. It is up to the application using the client's services to decide whether or not to set the criticality as to the support of paged results, and to handle a lack of support of this control on referral servers as appropriate based on the application. Additionally, the LDAP server does not ensure that the referral server supports the paged results control. Multiple lists can be returned to the client application, some not paged. It is the client application's decision as to how best to present this information to the end user. Possible solutions include:

- Combine all referral results before presenting to the end user
- Show multiple lists and the corresponding referral server host name
- Take no extra steps and show all results to the end user as they are returned from the server

The client application must turn off referrals to get one truly paged list, otherwise when chasing referrals with the paged results search control specified, unpredictable results might occur.

More information about the simple paged results search control, with control OID of 1.2.840.113556.1.4.319, can be found in RFC 2686 - LDAP Control Extension for Simple Paged Results Manipulation.

Chapter 2. LDAP utilities

The following section provides detailed documentation for the client utilities:

LDAPMODIFY, LDAPADD

LDAP modify-entry and LDAP add-entry tools

Synopsis

```
ldapmodify [-a] [-b] [-c] [-C charset] [-d debuglevel] [-D binddn]
[-f file] [-h ldaphost] [-K keyfile] [-m mechanism] [-M]
[-N certificatename] [-O hopcount] [-p ldapport] [-P keyfilepw]
[-r] [-R] [-v] [-V] [-w passwd] [-Z]

ldapadd [-b] [-c]  [-d debuglevel] [-D binddn]
[-f file] [-h ldaphost] [-K keyfile] [-M]
[-N certificatename] [-p ldapport] [-P keyfilepw]
[-r] [-R] [-v] [-V] [-w passwd] [-Z]
```

Description

ldapmodify is a command-line interface to the ldap_modify and ldap_add library calls. **ldapadd** is implemented as a renamed version of ldapmodify. When invoked as ldapadd, the **-a** (add new entry) flag is turned on automatically.

ldapmodify opens a connection to an LDAP server, and binds to the server. You can use **ldapmodify** to modify or add entries. The entry information is read from standard input or from file through the use of the **-f** option.

To display syntax help for **ldapmodify** or **ldapadd**, type

```
ldapmodify -?
```

or

```
ldapadd -?
```

Options

-a Add new entries. The default action for **ldapmodify** is to modify existing entries. If invoked as **ldapadd**, this flag is always set.

-b Assume that any values that start with a forward slash (/) are binary values and that the actual value is in a file whose path is specified in place of the valuer.

-c Continuous operation mode. Errors are reported, but **ldapmodify** continues with modifications. Otherwise the default action is to exit after reporting an error.

-C *charset*
Specifies that strings supplied as input to the **ldapmodify** and **ldapadd**

utilities are represented in a local character set as specified by charset, and must be converted to UTF-8. When the **ldapmodify** and **ldapadd** records are received from standard input, the specified charset value is used to convert the attribute values that are designated as strings that is, the attribute types are followed by a single colon. If the records are received from an LDIF file that contains a charset tag, the charset tag in the LDIF file overrides the charset value specified on the command-line. See "IANA character sets supported by platform" on page 185 for the specific charset values that are supported for each operating system platform. Note that the supported values for charset are the same values supported for the charset tag that is optionally defined in Version 1 LDIF files.

-d *debuglevel*
Set the LDAP debugging level to debuglevel.

-D *binddn*
Use ***binddn*** to bind to the LDAP directory. ***binddn*** is a string-represented DN (see Appendix B, "LDAP distinguished names" on page 181).

Note: **-D** *binddn* **-w** *passwd* does not call bind functions on superuser DNs.

-f file Read the entry modification information from an LDIF file instead of from standard input. If an ldif file is not specified, you must use standard input to specify the update records in ldif format.

-h *ldaphost*
Specify an alternate host on which the ldap server is running.

-K *keyfile*
Specify the name of the SSL key database file with default extension of **kdb**. If the key database file is not in the current directory, specify the fully-qualified key database filename. If a key database filename is not specified, this utility first looks for the presence of the SSL_KEYRING environment variable with an associated filename. If the SSL_KEYRING environment variable is not defined, the default keyring file is used, if present.

A default keyring file that is, ldapkey.kdb, and the associated password stashfile that is, ldapkey.sth, are installed in the /lib directory under LDAPHOME, where LDAPHOME is the path to the installed LDAP support. LDAPHOME varies by operating system platform:

- Windows - c:\Program Files\IBM\LDAP
- AIX - /usr/ldap
- Solaris - /opt/IBMldapc
- Linux - /usr/ldap
- HP - /usr/IBMldap

Note: This is the default install location. The actual LDAPHOME is determined during installation.

See 109 for more information about default key database files, and default CAs.

If a keyring database file cannot be located, a hard-coded set of default trusted certificate authority roots is used. The key database file typically contains one or more certificates of CAs that are trusted by the client. These types of X.509 certificates are also known as trusted roots. For more information on managing an SSL key database, see Chapter 4, "Using

GSK5IKM" on page 131. Also see the "SSL notes" on page 14 and "LDAP_SSL" on page 107 for more information about SSL and certificates.

This parameter effectively enables the **-Z** switch.

-m *mechanism*
Use ***mechanism*** to specify the Simple Authentication Security Layer (SASL) mechanism to be used to bind to the server. The ldap_sasl_bind_s() API is used. The **-m** parameter is ignored if **-V 2** is set. If **-m** is not specified, simple authentication is used.

-M Manage referral objects as regular entries.

-N *certificatename*
Specify the label associated with the client certificate in the key database file. If the LDAP server is configured to perform server authentication only, a client certificate is not required. If the LDAP server is configured to perform client and server Authentication, a client certificate might be required. ***certificatename*** is not required if a default certificate/private key pair has been designated as the default. Similarly, ***certificatename*** is not required if there is a single certificate/private key pair in the designated key database file. This parameter is ignored if neither **-Z** nor **-K** is specified.

-O *hopcount*
Specify ***hopcount*** to set the maximum number of hops that the client library takes when chasing referrals. The default hopcount is 10.

-p *ldapport*
Specify an alternate TCP port where the ldap server is listening. The default LDAP port is 389. If **-p** is not specified and **-Z** is specified, the default LDAP SSL port 636 is used.

-P *keyfilepw*
Specify the key database password. This password is required to access the encrypted information in the key database file, which might include one or more private keys. If a password stash file is associated with the key database file, the password is obtained from the password stash file, and the **-P** parameter is not required. This parameter is ignored if neither **-Z** nor **-K** is specified.

-r Replace existing values by default.

-R Specifies that referrals are not to be automatically followed.

-v Use verbose mode, with many diagnostics written to standard output.

-V Specifies the LDAP version to be used by **ldapmodify** when it binds to the LDAP server. By default, an LDAP V3 connection is established. To explicitly select LDAP V3, specify **-V 3**. Specify **-V 2** to run as an LDAP V2 application. An application, like **ldapmodify**, selects LDAP V3 as the preferred protocol by using ldap_init instead of ldap_open.

-w *passwd*
Use ***passwd*** as the password for authentication.

-Z Use a secure SSL connection to communicate with the LDAP server. The **-Z** option is only supported when the SSL componentry, as provided by the Tivoli® GSKit, is installed.

Input format

The contents of file (or standard input if no **-f** flag is given on the command line) must conform to the ldif format.

Alternative input format

An alternative input format is supported for compatibility with older versions of **ldapmodify**. This format consists of one or more entries separated by blank lines, where each entry looks like the following:

```
Distinguished Name (DN)

attr=value

[attr=value ...]
```

where `attr` is the name of the attribute and `value` is the value.

By default, values are added. If the **-r** command line flag is given, the default is to replace existing values with the new one. It is permissible for a given attribute to appear more than once, for example, to add more than one value for an attribute. Also note that you can use a trailing double back slash (\\ ) to continue values across lines and preserve new lines in the value itself. This is useful for modifying QUIPU iattr attributes among others.

`attr` must be preceded by a - to remove a value. The = and `value` must be omitted to remove an entire attribute.

`attr` must be preceded by a + to add a value in the presence of the **-r** flag.

Examples

Assuming that the file /tmp/entrymods exists and has the following contents:

```
dn: cn=Modify Me, o=University of Higher Learning, c=US

      changetype: modify

      replace: mail

      mail: modme@student.of.life.edu

      -

      add: title

      title: Grand Poobah

      -

      add: jpegPhoto

      jpegPhoto: /tmp/modme.jpeg

      -

      delete: description

      -
```

the command:

```
ldapmodify -b -r -f /tmp/entrymods
```

replaces the contents of the Modify Me entry's mail attribute with the value modme@student.of.life.edu, add a title of Grand Poobah, and the contents of the file /tmp/modme.jpeg as a jpegPhoto, and completely remove the description attribute. These same modifications can be performed using the older ldapmodify inout format:

```
cn=Modify Me, o=University of Higher Learning, c=US

      mail=modme@student.of.life.edu

      +title=Grand Poobah

      +jpegPhoto=/tmp/modme.jpeg

      -description
```

and the command:

```
ldapmodify -b -r -f /tmp/entrymods
```

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

```
dn: cn=John Doe, o=University of Higher Learning, c=US

      objectClass: person

      cn: John Doe

      cn: Johnny

      sn: Doe

      title: the world's most famous mythical person

      mail: johndoe@student.of.life.edu

      uid: jdoe
```

the command:

```
 ldapadd -f /tmp/entrymods
```

adds a new entry for John Doe, using the values from the file /tmp/newentry.

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

```
dn: cn=John Doe, o=University of Higher Learning, c=US

changetype: delete
```

the command:

```
 ldapmodify -f /tmp/entrymods
```

removes John Doe's entry.

Notes

If entry information is not supplied from file through the use of the **-f** option, the **ldapmodify** command waits to read entries from standard input. To break out of the wait, press **Ctrl+C** or **Ctrl+D**.

Diagnostics

Exit status is 0 if no errors occur. Errors result in a non-zero exit status and a diagnostic message being written to standard error.

SSL notes

To use the SSL-related functions associated with this utility, the SSL libraries and tools must be installed. The SSL libraries and tools are provided with the Tivoli Global Security Kit (GSKit), which includes RSA Security Inc. software.

Note: For information regarding the use of encryption by LDAP applications, including the LDAP sample programs, see "Usage" on page 111. This section describes the steps required to build the sample programs and your applications so they can use SSL encryption algorithms.

The content of a client's key database file is managed with the gsk5ikm utility. For more information on this Java™ utility, see Chapter 4, "Using GSK5IKM" on page 131. The gsk5ikm utility is used to define the set of trusted certification authorities (CAs) that are to be trusted by the client. By obtaining certificates from trusted CAs, storing them in the key database file, and marking them as trusted, you can establish a trust relationship with LDAP servers that use trusted certificates issued by one of the trusted CAs. The gsk5ikm utility can also be used to obtain a client certificate, so that client and server authentication can be performed.

If the LDAP servers accessed by the client use server authentication only, it is sufficient to define one or more trusted root certificates in the key database file. With server authentication, the client can be assured that the target LDAP server has been issued a certificate by one of the trusted CAs. In addition, all LDAP transactions that flow over the SSL connection with the server are encrypted including the LDAP credentials that are supplied on the ldap_bind or ldap_simple_bind_s (see "LDAP_BIND / UNBIND" on page 36). For example, if the LDAP server is using a high-assurance VeriSign certificate, you must obtain a CA certificate from VeriSign, import it into your key database file, and mark it as trusted. If the LDAP server is using a self-signed server certificate, the administrator of the LDAP server can supply you with a copy of the server's certificate request file. Import the certificate request file into your key database file and mark it as trusted.

If the LDAP servers accessed by the client use client and server authentication, it is necessary to:

- Define one or more trusted root certificates in the server's key database file. This allows the client to be assured that the target LDAP server has been issued a certificate by one of the trusted CAs. In addition, all LDAP transactions that flow over the SSL connection with the server are encrypted, including the LDAP credentials that are supplied on the ldap_bind or ldap_simple_bind_s (see "LDAP_BIND / UNBIND" on page 36).
- Create a key pair using gsk5ikm and request a client certificate from a CA. After receiving the signed certificate from the CA, store the certificate in the client key database file.

See also

ldapdelete, ldapmodrdn, ldapsearch, ldap, ldap_add, ldap_delete, ldap_modify, ldap_modrdn, ldap_ssl_init, ldif, ldap_dn

LDAPDELETE

LDAP delete-entry tool

Synopsis

```
ldapdelete [-b searchbase] [-c] [-C charset] [-d debuglevel]
[-D binddn] [-f file] [-h ldaphost] [-K keyfile]
[-m mechanism] [-M] [-N certificatename] [-O hopcount]
[-pldapport] [-P keyfilepw] [-R] [-v] [-V] [-w passwd] [-Z] [dn] ...
```

Description

ldapdelete is a command-line interface to the ldap_delete library call.

ldapdelete opens a connection to an LDAP server, binds, and deletes one or more entries. If one or more Distinguished Name (DN) arguments are provided, entries with those DNs are deleted. Each DN is a string-represented DN (see Appendix B, "LDAP distinguished names" on page 181). If no DN arguments are provided, a list of DNs is read from standard input, or from file if the **-f** flag is used.

To display syntax help for **ldapdelete**, type:

```
ldapdelete -?
```

Options

-b *searchbase*
: Use searchbase as the starting point for the search instead of the default. If **-b** is not specified, this utility examines the LDAP_BASEDN environment variable for a searchbase definition.

-c
: Continuous operation mode. Errors are reported, but **ldapdelete** continues with modifications. Otherwise he default action is to exit after reporting an error.

-C *charset*
: Specifies that the DNs supplied as input to the **ldapdelete** utility are represented in a local character set, as specified by charset. Use **-C** *charset* to override the default, where strings must be supplied in UTF-8. See "IANA character sets supported by platform" on page 185 for the specific charset values that are supported for each operating system platform. Note that the supported values for charset are the same values supported for the charset tag that is optionally defined in Version 1 LDIF files.

-d *debuglevel*
: Set the LDAP debugging level to debuglevel.

-dn
: Specifies one or more DN arguments. Each DN must be a string-represented DN. See Appendix B, "LDAP distinguished names" on page 181.

-D *binddn*
: Use ***binddn*** to bind to the LDAP directory. ***binddn*** is a string-represented DN. See Appendix B, "LDAP distinguished names" on page 181.

-f *file*
: Read a series of lines from file, performing one LDAP delete for each line in the file. Each line in the file must contain a single distinguished name.

-h *ldaphost*
: Specify an alternate host on which the ldap server is running.

-K *keyfile*

Specify the name of the SSL key database file with default extension of **kdb**. If the key database file is not in the current directory, specify the fully-qualified key database filename. If a key database filename is not specified, this utility first looks for the presence of the SSL_KEYRING environment variable with an associated filename. If the SSL_KEYRING environment variable is not defined, the default keyring file is used, if present.

A default keyring file that is, ldapkey.kdb, and the associated password stashfile that is, ldapkey.sth, are installed in the /lib directory under LDAPHOME, where LDAPHOME is the path to the installed LDAP support. LDAPHOME varies by operating system platform:

- Windows - c:\Program Files\IBM\LDAP
- AIX - /usr/ldap
- Solaris - /opt/IBMldapc
- Linux - /usr/ldap
- HP - /usr/IBMldap

Note: This is the default install location. The actual LDAPHOME is determined during installation.

See 109 for more information about default key database files, and default CAs.

If a keyring database file cannot be located, a hard-coded set of default trusted certificate authority roots is used. The key database file typically contains one or more certificates of CAs that are trusted by the client. These types of X.509 certificates are also known as trusted roots. For more information on managing an SSL key database, see Chapter 4, "Using GSK5IKM" on page 131. Also see the "SSL notes" on page 17 and "LDAP_SSL" on page 107 for more information about SSL and certificates.

This parameter effectively enables the **-Z** switch.

-m *mechanism*

Use ***mechanism*** to specify the Simple Authentication Security Layer (SASL) mechanism to be used to bind to the server. The ldap_sasl_bind_s() API is used. The **-m** parameter is ignored if **-V 2** is set. If **-m** is not specified, simple authentication is used.

-M Manage referral objects as regular entries.

-N *certificatename*

Specify the label associated with the client certificate in the key database file. If the LDAP server is configured to perform server authentication only, a client certificate is not required. If the LDAP server is configured to perform client and server Authentication, a client certificate might be required. ***certificatename*** is not required if a default certificate/private key pair has been designated as the default. Similarly, ***certificatename*** is not required if there is a single certificate/private key pair in the designated key database file. This parameter is ignored if neither **-Z** nor **-K** is specified.

-O *hopcount*

Specify ***hopcount*** to set the maximum number of hops that the client library takes when chasing referrals. The default hopcount is 10.

-p *ldapport*
Specify an alternate TCP port where the ldap server is listening. The default LDAP port is 389. If **-p** is not specified and **-Z** is specified, the default LDAP SSL port 636 is used.

-P *keyfilepw*
Specify the key database password. This password is required to access the encrypted information in the key database file, which can include one or more private keys. If a password stash file is associated with the key database file, the password is obtained from the password stash file, and the **-P** parameter is not required. This parameter is ignored if neither **-Z** nor **-K** is specified.

-R Specifies that referrals are not to be automatically followed.

-v Use verbose mode, with many diagnostics written to standard output.

-V Specifies the LDAP version to be used by **ldapdelete** when it binds to the LDAP server. By default, an LDAP V3 connection is established. To explicitly select LDAP V3, specify **-V 3**. Specify **-V 2** to run as an LDAP V2 application. An application, like **ldapdelete**, selects LDAP V3 as the preferred protocol by using ldap_init instead of ldap_open.

-w *passwd*
Use ***passwd*** as the password for authentication.

-Z Use a secure SSL connection to communicate with the LDAP server. The **-Z** option is only supported when the SSL componentry, as provided by the GSKit, is installed.

Examples

The following command,

```
ldapdelete "cn=Delete Me, o=University of Life, c=US"
```

attempts to delete the entry named with commonName Delete Me directly below the University of Life organizational entry. It might be necessary to supply a ***binddn*** and ***passwd*** for deletion to be allowed (see the **-D** and **-w** options).

Notes

If no DN arguments are provided, the **ldapdelete** command waits to read a list of DNs from standard input. To break out of the wait, press **Ctrl+C** or **Ctrl+D**.

Diagnostics

Exit status is 0 if no errors occur. Errors result in a non-zero exit status and a diagnostic message being written to standard error.

SSL notes

See “SSL notes” on page 14.

See also

ldapadd, ldapmodify, ldapmodrdn, ldapsearch, ldap, ldap_add, ldap_delete, ldap_modify, ldap_modrdn, ldap_ssl_init, ldif, ldap_dn

LDAPMODRDN

LDAP modify-entry RDN tool

Synopsis

```
ldapmodrdn [-c] [-C charset] [-d debuglevel]
[-D binddn] [-f file] [-h ldaphost] [-K keyfile]
[-m mechanism] [-M] [-N certificatename]
[-O hopcount] [-p ldapport] [-P keyfilepw] [-r]
[-R] [-v] [-V] [-w passwd] [-Z] [dnrdn] ...
```

Description

ldapmodrdn is a command-line interface to the ldap_modrdn library call.

ldapmodrdn opens a connection to an LDAP server, binds, and modifies the RDN of entries. The entry information is read from standard input, from file through the use of the **- f** option, or from the command-line pair dn and rdn.

See LDAP Distinguished Names for information about RDNs (Relative Distinguished Names) and DNs (Distinguished Names).

To display syntax help for **ldapmodrdn**, type:

```
ldapmodrdn -?
```

Options

-c Continuous operation mode. Errors are reported, but **ldapmodrdn** continues with modifications. Otherwise the default action is to exit after reporting an error.

-C *charset*
Specifies that the strings supplied as input to the **ldapmodrdn** utility are represented in a local character set, as specified by charset. Use **-C** *charset* to override the default, where strings must be supplied in UTF-8. See "IANA character sets supported by platform" on page 185 for the specific charset values that are supported for each operating system platform. Note that the supported values for charset are the same values supported for the charset tag that is optionally defined in Version 1 LDIF files.

-d *debuglevel*
Set the LDAP debugging level to debuglevel.

-D *binddn*
Use ***binddn*** to bind to the LDAP directory. binddn must be a string-represented DN (see Appendix B, "LDAP distinguished names" on page 181).

-f *file* Read the entry modification information from file instead of from standard input or the command-line (by specifying rdn and newrdn). Standard input can be supplied from a file, as well (< file).

-h *ldaphost*
Specify an alternate host on which the ldap server is running.

-K *keyfile*
Specify the name of the SSL key database file (with default extension of kdb). If the key database file is not in the current directory, specify the fully-qualified key database filename. If a key database filename is not specified, this utility first looks for the presence of the SSL_KEYRING environment variable with an associated filename. If the SSL_KEYRING environment variable is not defined, the default keyring file is used, if present.

A default keyring file (that is, ldapkey.kdb) and the associated password stashfile (that is, ldapkey.sth) are installed in the /lib directory under LDAPHOME, where LDAPHOME is the path to the installed LDAP support. LDAPHOME varies by operating system platform:

- Windows - c:\Program Files\IBM\LDAP
- AIX - /usr/ldap
- Solaris - /opt/IBMldapc
- Linux - /usr/ldap
- HP - /usr/IBMldap

Note: This is the default install location. The actual LDAPHOME is determined during installation.

See 109 for more information about default key database files, and default CAs.

If a keyring database file cannot be located, a hard-coded set of default trusted certificate authority roots is used. The key database file typically contains one or more certificates of CAs that are trusted by the client. These types of X.509 certificates are also known as trusted roots. For more information on managing an SSL key database, see Chapter 4, "Using GSK5IKM" on page 131. Also see the "SSL notes" on page 21 and "LDAP_SSL" on page 107 for more information about SSL and certificates.

This parameter effectively enables the **-Z** switch.

-m *mechanism*
Use ***mechanism*** to specify the SASL mechanism to be used to bind to the server. The ldap_sasl_bind_s() API is used. The **-m** parameter is ignored if **-V 2** is set. If **-m** is not specified, simple authentication is used.

-M Manage referral objects as regular entries.

-N *certificatename*
Specify the label associated with the client certificate in the key database file. Note that if the LDAP server is configured to perform server authentication only, a client certificate is not required. If the LDAP server is configured to perform client and server Authentication, a client certificate might be required. ***certificatename*** is not required if a default certificate/private key pair has been designated as the default. Similarly, ***certificatename*** is not required if there is a single certificate/private key pair in the designated key database file. This parameter is ignored if neither **-Z** nor **-K** is specified.

-O *hopcount*
Specify ***hopcount*** to set the maximum number of hops that the client library takes when chasing referrals. The default hopcount is 10.

-p *ldapport*
Specify an alternate TCP port where the ldap server is listening. The default LDAP port is 389. If not specified and -Z is specified, the default LDAP SSL port 636 is used.

-P *keyfilepw*
Specify the key database password. This password is required to access the encrypted information in the key database file, which can include one or more private keys. If a password stash file is associated with the key

database file, the password is obtained from the password stash file, and the **-P** parameter is not required. This parameter is ignored if neither **-Z** nor **-K** is specified.

-r Remove old RDN values from the entry. Default action is to keep old values.

-R Specifies that referrals are not to be automatically followed.

-v Use verbose mode, with many diagnostics written to standard output.

-V Specifies the LDAP version to be used by **ldapmodrdn** when it binds to the LDAP server. By default, an LDAP V3 connection is established. To explicitly select LDAP V3, specify **-V 3**. Specify **-V 2** to run as an LDAP V2 application. An application, like **ldapmodrdn**, selects LDAP V3 as the preferred protocol by using ldap_init instead of ldap_open.

-w *passwd*
Use ***passwd*** as the password for authentication.

-Z Use a secure SSL connection to communicate with the LDAP server. The **-Z** option is only supported when the SSL componentry, as provided by the Tivoli GSKit, is installed.

dn rdn

Input format

If the command-line arguments *dn* and *rdn* are given, *rdn* replaces the RDN of the entry specified by the DN, *dn*. Otherwise, the contents of file (or standard input if no **- f** flag is given) consist of one or more entries:

```
Distinguished Name (DN)

Relative Distinguished Name (RDN)
```

One or more blank lines can be used to separate each DN and RDN pair.

Examples

Assuming that the file /tmp/entrymods exists and has the contents:

```
cn=Modify Me, o=University of Life, c=US
cn=The New Me
```

the command:

```
ldapmodrdn -r -f /tmp/entrymods
```

changes the RDN of the `Modify Me` entry from `Modify Me` to `The New Me` and the old cn, `Modify Me` is removed.

Notes

If entry information is not supplied from file through the use of the **-f** option or from the command-line pair *dn* and *rdn*, the **ldapmodrdn** command waits to read entries from standard input. To break out of the wait, press **Ctrl+C** or **Ctrl+D**.

Diagnostics

Exit status is 0 if no errors occur. Errors result in a non-zero exit status and a diagnostic message being written to standard error.

SSL notes

See “SSL notes” on page 14.

See also

ldapadd, ldapdelete, ldapsearch, ldap, ldap_add, ldap_delete, ldap_modify, ldap_modrdn, ldap_ssl_init, ldif, ldap_dn

LDAPSEARCH

LDAP search tool and sample program

Synopsis

```
ldapsearch [-a deref] [-A] [-b searchbase] [-B] [-C charset]
[-d debuglevel] [-D binddn] [-f file] [filter] [-F sep]
[-h ldaphost] [-K keyfile] [-l timelimit] [-L] [-m mechanism]
[-M] [-N certificatename] [-o attributename] [-O hopcount]
[-p ldapport] [-P keyfilepw] [-q page size] [-R] [-s scope ]
[-t] [-T seconds] [-v] [-V] [-w bindpasswd] [-z sizelimit] [-Z]
[attrs...]
```

Description

ldapsearch is a command-line interface to the ldap_search library call.

ldapsearch opens a connection to an LDAP server, binds, and performs a search using the filter. The filter must conform to the string representation for LDAP filters (see ldap_search for more information on filters).

If **ldapsearch** finds one or more entries, the attributes specified by attrs are retrieved and the entries and values are printed to standard output. If no attrs are listed, all attributes are returned.

To display syntax help for **ldapsearch**, type `ldapsearch -?`

Options

-a deref
: Specify how aliases dereferencing is done. **deref** must 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 just want to see if an attribute is present in an entry and are not interested in the specific values.

-b searchbase
: Use searchbase as the starting point for the search instead of the default. If **-b** is not specified, this utility examines the LDAP_BASEDN environment variable for a searchbase definition. If neither is set, the default base is set to "".

-B
: Do not suppress display of non-ASCII values. This is useful when dealing with values that appear in alternate characters sets such as ISO-8859.1. This option is implied by the **-L** option.

-C charset

Specifies that strings supplied as input to the **ldapsearch** utility are represented in a local character set, as specified by charset. String input includes the filter, the bind DN and the base DN. Similarly, when displaying data, **ldapsearch** converts data received from the LDAP server to the specified character set. Use **-C charset** to override the default, where strings must be supplied in UTF-8. Also, if the **-C** option and the **-L** option are both specified, input is assumed to be in the specified character set, but output from **ldapsearch** is always preserved in its UTF-8 representation, or a base-64 encoded representation of the data when non-printable characters are detected. This is the case because standard LDIF files contain UTF-8 or base-64-encoded UTF-8 representations of string data only. See "IANA character sets supported by platform" on page 185 for the specific charset values that are supported for each operating system platform. Note that the supported values for charset are the same values supported for the charset tag that is optionally defined in Version 1 LDIF files.

-d debuglevel

Set the LDAP debugging level to debuglevel.

-D binddn

Use **binddn** to bind to the LDAP directory. **binddn** must be a string-represented DN. See Appendix B, "LDAP distinguished names" on page 181 for more information.

-f file Read a series of lines from file, performing one LDAP search for each line. In this case, the filter given on the command line is treated as a pattern where the first occurrence of %s is replaced with a line from file. If file is a single hyphen (-) character, then the lines are read from standard input.

filter Specifies a string representation of the filter to apply in the search. Simple filters can be specified as attributetype=attributevalue. More complex filters are specified using a prefix notation according to the following Backus Naur Form (BNF):

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

The '~=' construct is used to specify approximate matching. The representation for <*attributetype*> and <*attributevalue*> are as described in "RFC 2252, LDAP V3 Attribute Syntax Definitions". In addition, <*attributevalue*> can be a single * to achieve an attribute existence test, or can contain text and asterisks (*) interspersed to achieve substring matching.

For example, the filter "mail=*" finds any entries that have a mail attribute. The filter "mail=*@student.of.life.edu" finds any entries that have a mail attribute ending in the specified string. To put parentheses in a filter, escape them with a backslash (\ ) character.

Note: A filter like `"cn=Bob *"`, where there is a space between Bob and the asterisk (*), matches "Bob Carter" but not "Bobby Carter" in IBM

Directory. The space between ″Bob″ and the wildcard character (*) affects the outcome of a search using filters.

See ″RFC 2254, A String Representation of LDAP Search Filters″ for a more complete description of allowable filters.

-F sep Use sep as the field separator between attribute names and values. The default separator is equals (=), unless the **-L** flag has been specified, in which case this option is ignored.

-h ldaphost
Specify an alternate host on which the ldap server is running.

-K keyfile
Specify the name of the SSL key database file with default extension of .kdb. If the key database file is not in the current directory, specify the fully-qualified key database filename. If a key database filename is not specified, this utility first looks for the presence of the SSL_KEYRING environment variable with an associated filename. If the SSL_KEYRING environment variable is not defined, the default keyring file is used, if present.

A default keyring file (ldapkey.kdb) and the associated password stashfile (ldapkey.sth) are installed in the /lib directory under LDAPHOME, where LDAPHOME is the path to the installed LDAP support. LDAPHOME varies by operating system platform:

- Windows - c:\Program Files\IBM\LDAP
- AIX - /usr/ldap
- Solaris - /opt/IBMldapc
- Linux - /usr/ldap
- HP - /usr/IBMldap

Note: This is the default install location. The actual LDAPHOME is determined during installation.

See 109 for more information about default key database files, and default CAs.

If a keyring database file cannot be located, a hard-coded set of default trusted certificate authority roots is used. The key database file typically contains one or more certificates of CAs that are trusted by the client. These types of X.509 certificates are also known as trusted roots. For more information on managing an SSL key database, see Chapter 4, “Using GSK5IKM” on page 131. Also see “SSL notes” on page 29 and “LDAP_SSL” on page 107 for more information about SSL and certificates.

This parameter effectively enables the **-Z** switch.

-l timelimit
Wait at most *timelimit* seconds for a search to complete.

-L Display search results in ldif format. This option also turns on the **-B** option, and causes the **-F** option to be ignored.

-m mechanism
Use **mechanism** to specify the SASL mechanism to be used to bind to the server. The ldap_sasl_bind_s() API is used. The **-m** parameter is ignored if **-V 2** is set. If **-m** is not specified, simple authentication is used.

-M Manage referral objects as regular entries.

-N certificatename
Specify the label associated with the client certificate in the key database file.

Note: If the LDAP server is configured to perform server authentication only, a client certificate is not required. If the LDAP server is configured to perform client and server Authentication, a client certificate might be required. *certificatename* is not required if a default certificate/private key pair has been designated as the default. Similarly, *certificatename* is not required if there is a single certificate/private key pair in the designated key database file. This parameter is ignored if neither **-Z** nor **-K** is specified.

-o attributename
To specify an attribute to use for sort criteria of search results, you can use the -o (order) parameter. You can use multiple -o parameters to further define the sort order. In the following example, the search results are sorted first by surname (sn), then by given name (givenname), with the given name being sorted in reverse (descending) order as specified by the prefixed minus sign (-):

```
-o sn -o -givenname
```

Thus, the syntax of the sort parameter is as follows:

```
[-]<attribute name>[:<matching rule OID>]
```

where

- `attribute name` is the name of the attribute you want to sort by.
- `matching rule OID` is the optional OID of a matching rule that you want to use for sorting.
- The minus sign (-) indicates that the results must be sorted in reverse order.
- The criticality is always critical.

The default ldapsearch operation is not to sort the returned results.

-O hopcount
Specify **hopcount** to set the maximum number of hops that the client library takes when chasing referrals. The default hopcount is 10.

-p ldapport
Specify an alternate TCP port where the ldap server is listening. The default LDAP port is 389. If an alternate TCP port is not specified, and -Z is specified, the default LDAP SSL port 636 is used.

-P keyfilepw
Specify the key database password. This password is required to access the encrypted information in the key database file (which can include one or more private keys). If a password stash file is associated with the key database file, the password is obtained from the password stash file, and the **-P** parameter is not required. This parameter is ignored if neither **-Z** nor **-K** is specified.

-q page size
To specify paging of search results, two new parameters can be used: -q (query page size), and -T (time between searches, in seconds). In the following example, the search results return a page (25 entries) at a time,

every 15 seconds, until all the results for that search are returned. The ldapsearch client handles all connection continuation for each paged results request for the life of the search operation.

```
-q 25  -T 15
```

If the -v (verbose) parameter is specified, ldapsearch lists how many entries have been returned so far after each page of entries returned from the server, for example, **30 total entries have been returned.** Multiple -q parameters are enabled such that you can specify different page sizes throughout the life of a single search operation. In the following example, the first page is 15 entries, the second page is 20 entries, and the third parameter ends the paged result/search operation:

```
-q 15 -q 20 -q 0
```

In the following example, the first page is 15 entries, and all the rest of the pages are 20 entries, continuing with the last specified -q value until the search operation completes:

```
-q 15 -q 20
```

The default ldapsearch operation is to return all entries in a single request. No paging is done for the default ldapsearch operation.

-R Specifies that referrals are not to be automatically followed.

-s scope
Specify the scope of the search. **scope** must be one of **base**, **one**, or **sub** to specify a base object, one-level, or subtree search. The default is **sub**.

-t Write retrieved values to a set of temporary files. This is useful for dealing with non-ASCII values such as jpegPhoto or audio.

-T seconds
Time between searches (in seconds). The -T option is only supported when the -q option is specified.

-v Use verbose mode, with many diagnostics written to standard output.

-V Specifies the LDAP version to be used by ldapmodify when it binds to the LDAP server. By default, an LDAP V3 connection is established. To explicitly select LDAP V3, specify `-V 3`. Specify `-V 2` to run as an LDAP V2 application. An application, like ldapmodify, selects LDAP V3 as the preferred protocol by using ldap_init instead of ldap_open.

-w passwd
Use **passwd** as the password for authentication.

-z sizelimit
Limit the results of the search to at most *sizelimit* entries. This makes it possible to place an upper bound on the number of entries that are returned for a search operation.

-Z Use a secure SSL connection to communicate with the LDAP server. The **-Z** option is only supported when the SSL componentry, as provided by the Tivoli GSKit, is installed.

Output format

If one or more entries are found, each entry is written to standard output in the following 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 equals (=) character. 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

The following command:

```
ldapsearch "cn=john doe" cn telephoneNumber
```

performs a subtree search (using the default search base) for entries with a commonName of john doe. The commonName and telephoneNumber values are retrieved and printed to standard output. If two entries are found, the output might look something like this:

```
  cn=John E Doe, ou="College of Literature, Science, and the Arts",
ou=Students, ou=People, o=University of Higher Learning, c=US

  cn=John Doe

  cn=John Edward Doe

  cn=John E Doe 1

  cn=John E Doe

  telephoneNumber=+1 313 555-5432



  cn=John B Doe, ou=Information Technology Division,
ou=Faculty and Staff, ou=People, o=University of Higher Learning, c=US

  cn=John Doe

  cn=John B Doe 1

  cn=John B Doe

  telephoneNumber=+1 313 555-1111
```

The command

```
ldapsearch -t "uid=jed" jpegPhoto audio
```

performs a subtree search using the default search base for entries with user ID of ″jed″. 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=John E Doe, ou=Information Technology Division,

  ou=Faculty and Staff,
```

```
  ou=People, o=University of Higher Learning, c=US

  audio=/tmp/ldapsearch-audio-a19924

  jpegPhoto=/tmp/ldapsearch-jpegPhoto-a19924
```

The command

```
ldapsearch -L -s one -b "c=US" "o=university*" o description
```

performs a one-level search at the c=US level for all organizations whose organizationName begins with `university`. Search results are displayed in the LDIF format (see LDAP Data Interchange Format). The organizationName and description attribute values are retrieved and printed to standard output, resulting in output similar to the following:

```
dn: o=University of Neptune, c=US

  o: University of Neptune

  description: Preparing Neptune for a brave new tomorrow

  description: leaf node only


  dn: o=University of Saturn at Pluto, c=US

  o: University of Saturn at Pluto

  description: No personnel information

  description: Institution of education and research


  dn: o=University of Saturn at Venus, c=US

  o: University of Saturn at Venus

  o: USV

  o: SU/Venus

  o: SU-Venus

  description: Institute for Higher Learning and Research


  dn: o=University of Jupiter, c=US

  o: University of Jupiter

  o: UJu

  description: Shaper of young minds


 ...
```

The following command is a complex search:

```
ldapsearch -D cn=root -w password -b basetosearch
        "(&(modifytimestamp >= 20001228080000)
        (modifytimestamp <= 20001228120000)"
```

This example searches for all entries that were modified between 8 am and 12 noon on December 28, 2000.

Note: Be sure to include the quotation marks (″) in the command. If the quotation marks are not present in the command, the command can fail.

The command

```
ldapsearch -b "o=Fictional Team, c=US" -o sn "givenname=B*" cn
```

performs a subtree search on all entries that exist in the Fictional Team organization (which is under the c=US level), whose given name (givenname) begins with B. Search results are sorted based on surname (sn). If two entries are found the output might look like this:

```
cn=Bret Fiction, ou=MVP, o=Fictional Team, c=US
cn=Bret Fiction

cn=Bubba Ford, ou=Offense, o=Fictional Team, c=US
cn=Bubba Ford
```

The command

```
ldapsearch -v -b "o=Fictional Team, c=US" -o sn -q 5 -T 3 "givenname=*" cn
```

performs a subtree search on all entries that exist in the Fictional Team organization (which is under the c=US level), whose given name (givenname) begins with anything (even NULL). Search results are sorted based on surname (sn) one page at a time (5 entries), with a 3 second wait between pages. If six entries are found, the output might look like the following:

```
cn=Gilbert Blue, ou=Defense, o=Fictional Team, c=US
cn=Gilbert Blue

cn=Larry Butler, ou=Defense, o=Fictional Team, c=US
cn=Larry Butler

cn=Bret Fiction, ou=MVP, o=Fictional Team, c=US
cn=Bret Fiction

cn=Bubba Ford, ou=Offense, o=Fictional Team, c=US
cn=Bubba Ford

cn=Bill Schooner, ou=Offense, o=Fictional Team, c=US
cn=Bill Schooner
5 total entries have been returned
5 matches

cn=Frank White, ou=Offense, o=Fictional Team, c=US
cn=Frank White
6 total entries have been returned
1 matches
```

Note: The verbose (-v) option displays the running total of entries returned and the number of matches for the current page.

Diagnostics

Exit status is 0 if no errors occur. Errors result in a non-zero exit status and a diagnostic message being written to standard error.

SSL notes

See “SSL notes” on page 14.

See also

ldapadd, ldapdelete, ldap, ldap_modify, ldap_modrdn, ldap_ssl_init, ldif, ldap_dn, ldap_sort, ldap_parse_results

Chapter 3. API categories

The following sets of APIs are supported by the IBM Directory:

LDAP_ABANDON

ldap_abandon

ldap_abandon_ext

Purpose

Abandon an LDAP operation in progress.

Synopsis

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

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

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

Input parameters

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

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

serverctrls
Specifies a list of LDAP server controls. This parameter can be set to NULL. See "LDAP controls" on page 45 for more information about server controls.

clientctrls
Specifies a list of LDAP client controls. This parameter can be set to NULL. See "LDAP controls" on page 45 for more information about client controls.

Usage

The ldap_abandon() and ldap_abandon_ext() APIs are used to abandon or cancel an LDAP operation in progress. The msgid passed must be the message ID of an outstanding LDAP operation, as returned by a call to an asynchronous LDAP operation such as ldap_search(), ldap_modify(), and so forth.

Both APIs check to see if the result of the operation has already been returned by the server. If the result of the operation has been returned, both APIs delete the result of the operation from the queue of pending messages. If not, both APIs send an LDAP abandon operation to the LDAP server.

The result of an abandoned operation is not returned from a future call to ldap_result().

The ldap_abandon_ext returns the constant LDAP_SUCCESS if the abandon was successful, or another LDAP error code if not. The ldap_abandon API returns zero if the abandon was successful, -1 if unsuccessful, and does not support LDAP V3 server controls or client controls.

Errors

ldap_abandon() returns 0 if the operation is successful, -1 if unsuccessful, setting ld_errno appropriately. See "LDAP_ERROR" on page 50 for details.
ldap_abandon_ext() returns LDAP_SUCCESS if successful and returns an LDAP error code if unsuccessful.

See also

ldap, ldap_result, ldap_error

LDAP_ADD

ldap_add

ldap_add_s

ldap_add_ext

ldap_add_ext_s

Purpose

Perform an LDAP operation to add an entry.

Synopsis

```
#include <ldap.h>


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

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

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

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

Input parameters

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

dn The DN of the entry to add.

attrs The entry's attributes, specified using the LDAPMod structure, as defined for ldap_modify(). The mod_type and mod_vals fields must be filled in. The mod_op field is ignored unless ORed with the constant LDAP_MOD_BVALUES. In this case, the mod_op field is used to select the mod_bvalues case of the mod_vals union.

serverctrls
Specifies a list of LDAP server controls. This parameter can be set to NULL. See "LDAP controls" on page 45 for more information about server controls.

clientctrls
Specifies a list of LDAP client controls. This parameter can be set to NULL. See "LDAP controls" on page 45 for more information about client controls.

Output parameters

msgidp
This result parameter is set to the message ID of the request if the ldap_add_ext() call succeeds.

Usage

The ldap_add() and associated APIs are used to perform an LDAP add operation. They take **dn**, the DN of the entry to add, and **attrs**, a NULL-terminated array of the entry's attributes. The LDAPMod structure (as defined for ldap_modify()) is used to represent attributes, with the mod_type and mod_values fields being filled in and used as described for ldap_modify(). The mod_op field is ignored unless ORed with the constant LDAP_MOD_BVALUES. In this case, the mod_op field is used to select the mod_bvalues case of the mod_vals union.

Note: All entries except those specified by the last component in the given DN must already exist.

The ldap_add_ext() API initiates an asynchronous add operation and returns the constant LDAP_SUCCESS if the request was successfully sent, or another LDAP error code if not. If successful, ldap_add_ext() places the message ID of the request in *msgidp. A subsequent call to ldap_result() can be used to obtain the result of the operation. Once the operation has completed, ldap_result() returns a result that contains the status of the operation (in the form of an error code). The error code indicates if the operation completed successfully. The ldap_parse_result() API is used to check the error code in the result.

Similarly, the ldap_add() API initiates an asynchronous add operation and returns the message ID of the operation initiated. A subsequent call to ldap_result(), can be used to obtain the result of the add. In case of error, ldap_add() returns -1, setting the session error parameters in the LDAP structure appropriately, which can be obtained by using ldap_get_errno().

See "LDAP_ERROR" on page 50 for more details.

The ldap_add_ext() and ldap_add_ext_s() APIs both support LDAP V3 server controls and client controls.

Errors

ldap_add() returns -1 in case of error initiating the request. ldap_add_s() and ldap_add_ext_s returns an LDAP error code directly; LDAP_SUCCESS if the call was successful, an LDAP error if the call was unsuccessful.

See also

ldap, ldap_modify

LDAP_FIRST_ATTRIBUTE

ldap_count_attributes
ldap_first_attribute
ldap_next_attribute

Purpose

Step through LDAP entry attributes.

Synopsis

```
#include <ldap.h>


int ldap_count_attributes(
       LDAP           *ld,
       LDAPMessage    *entry)

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

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

Input parameters

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

Output parameters

berptr

This is an output parameter returned from ldap_first_attribute(), which returns a pointer to a BerElement that has been allocated to keep track of current position. It is an input and output parameter for subsequent calls to ldap_next_attribute(), where it specifies a pointer to a BerElement which was allocated by the previous call to ldap_first_attribute(). The BerElement structure is opaque to the application.

Usage

The ldap_count_attributes() routine returns a count of the number of attributes in an LDAP entry. If a NULL entry is returned from ldap_first_entry() or ldap_next_entry(), and is passed as input to ldap_count_attributes(), a -1 is returned.

The ldap_first_attribute() and ldap_next_attribute() routines are used to step through the attributes in an LDAP entry.

ldap_first_attribute() takes an entry as returned by ldap_first_entry() or ldap_next_entry() and returns a pointer to a buffer containing the first attribute type in the entry.

The pointer returned by ldap_first_attribute in berptr must be passed to subsequent calls to ldap_next_attribute and is used to step through the entry's attributes. When there are no attributes left to be retrieved, ldap_next_attribute() returns NULL, sets the error code to LDAP_SUCCESS and releases the memory allocated for the BerElement buffer. If an error occurs, NULL is returned and an error code is set.

Therefore, when NULL is returned, the ldap_get_errno() API must be used to determine whether or not an error has occurred.

If the caller fails to call ldap_next_attribute() a sufficient number of times to exhaust the list of attributes, the caller is responsible for freeing the BerElement pointed to by berptr when it is no longer needed by calling ldap_ber_free().

The attribute names returned by ldap_first_attribute() are suitable for inclusion in a call to ldap_get_values().

ldap_next_attribute() returns a string that contains the name of the next type in the entry. This string must be freed using ldap_memfree() when its use is completed.

The attribute names returned by ldap_next_attribute() are suitable for inclusion in a call to ldap_get_values() to retrieve the attribute's values.

Errors

If the ldap_first_attribute() call results in an error, then NULL is returned, the error code is set and the memory pointed to by berptr is automatically freed and set to NULL.

If the ldap_next_attribute() call results in an error, NULL is returned, the error code is then set and the memory pointed to by berptr must also be freed calling ldap_ber_free()

The ldap_get_errno() API can be used to obtain the error code. See "LDAP_ERROR" on page 50 for a description of possible error codes.

Notes

The ldap_first_attribute() and ldap_next_attribute routines allocate memory that might need to be freed by the caller through ldap_memfree.

See also

ldap, ldap_first_entry, ldap_get_values, ldap_memfree, ldap_error

---

LDAP_BIND / UNBIND

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

Purpose

LDAP routines for binding and unbinding.

Note: For IBM Directory Server Version 4.1, Kerberos 1.2 is used on the AIX operating systems. For IBM Directory Server Version 4.1, Kerberos 1.1 is

used on the Windows NT and Windows 2000 operating systems. IBM Directory Server version 4.1 does not support Kerberos authentication on the Solaris or HP operating systems.

Synopsis

```
#include <ldap.h>


int ldap_sasl_bind(
       LDAP            *ld,
       const char      *dn,
       const char      *mechanism,
       const struct berval *cred,
       LDAPControl     **servctrls,
       LDAPControl     **clientctrls,
       int             *msgidp)

int ldap_sasl_bind_s(
       LDAP            *ld,
       const char      *dn,
       const char      *mechanism,
       const struct berval *cred,
       LDAPControl     **servctrls,
       LDAPControl     **clientctrls,
       struct berval   **servercredp)

int ldap_simple_bind(
       LDAP            *ld,
       const char      *dn,
       const char      *passwd)


int ldap_simple_bind_s(
       LDAP            *ld,
       const char      *dn,
       const char      *passwd)

int ldap_unbind(
       LDAP            *ld)

int ldap_unbind_s(
       LDAP            *ld)

int ldap_unbind_ext(
       LDAP            *ld,
       LDAPControl     **servctrls,
       LDAPControl     **clientctrls)

void ldap_set_rebind_proc(
       LDAP            *ld,
       LDAPRebindProc  rebindproc)

int ldap_bind(
       LDAP            *ld,
       const char      *dn,
       const char      *cred,
       int             method)

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

Input parameters

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

dn Specifies the Distinguished Name (DN) of the entry to bind as.

cred Specifies the credentials with which to authenticate. Arbitrary credentials can be passed using this parameter. In most cases, this is the user's password. When using a Simple Authentication Security Layer (SASL) bind, the format and content of the credentials depends on the setting of the mechanism parameter.

mechanism

Although a variety of mechanisms have been IANA (Internet Assigned Numbers Authority) registered, the only native mechanisms supported by the LDAP library at this time are:

- LDAP_MECHANISM_EXTERNAL mechanism, represented by the string `EXTERNAL`.
- LDAP_MECHANISM_CRAMMD5 mechanism, represented by the string `cram-md5`.
- LDAP_MECHANISM_GSSAPI mechanism, represented by the string `GSSAPI`.

The LDAP_MECHANISM_EXTERNAL mechanism indicates to the server that information external to SASL must be used to determine whether the client is authorized to authenticate. For this implementation, the system providing the external information must be SSL. For example, if the client sets DN and credential to NULL (the value of the pointers must be NULL), with mechanism set to LDAP_MECHANISM_EXTERNAL, the client is requesting that the server use the strongly authenticated identity from the client's X.509 certificate that was used to authenticate the client to the server during the SSL handshake. The server can then use the strongly authenticated identity to access the directory.

The LDAP_MECHANISM_CRAMMD5 mechanism is used to authenticate your ID and password with the server using a challenge/response protocol that protects the clear-text password over the wire. This mechanism is useful only when the LDAP server can retrieve the user's password. If the password is stored in a hashed form, for example, crypt or SHA, then authentication using the **cram-md5** mechanism fails.

The LDAP_MECHANISM_GSSAPI mechanism is used to enable Kerberos authentication. In Kerberos authentication, a client presents valid credentials obtained from a Kerberos key distribution center (KDC) to an application server. The server decrypts and verifies the credentials using its service key.

See "LDAP_PLUGIN_REGISTRATION" on page 79 for more information about using LDAP client plug-ins. See Chapter 7, "LDAP client plug-in programming reference" on page 169 for more information about developing an LDAP client plug-in.

method

Selects the authentication method to use. Specify LDAP_AUTH_SIMPLE for simple authentication or LDAP_AUTH_SASL for SASL bind. Note that use of the ldap_bind and ldap_bind_s APIs is deprecated.

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

serverctrls
Specifies a list of LDAP server controls. See “LDAP controls” on page 45 for more information about server controls.

clientctrls
Specifies a list of LDAP client controls. See “LDAP controls” on page 45 for more information about client controls.

rebindproc
Specifies the entry-point of a routine that is called to obtain bind credentials used when a new server is contacted following an LDAP referral.

Output parameters

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

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

Usage

These routines provide various interfaces to the LDAP bind operation. After using ldap_init, ldap_ssl_init or ldap_open to create an LDAP handle, a bind can be performed before other operations are attempted over the connection. Both synchronous and asynchronous versions of each variant of the bind call are provided.

A bind is optional when communicating with an LDAP server that supports the LDAP V3 protocol. The absence of a bind is interpreted by the LDAP V3 server as a request for unauthenticated access. A bind is required by LDAP servers that only support the LDAP V2 protocol.

The ldap_simple_bind() and ldap_simple_bind_s() APIs provide simple authentication, using a user ID or **dn** and a password passed in clear-text to the LDAP API.

The ldap_bind() and ldap_bind_s() provide general authentication routines, where an authentication method can be chosen. In this toolkit, method must be set to LDAP_AUTH_SIMPLE. Because the use of these two APIs is deprecated, ldap_simple_bind and ldap_simple_bind_s must be used instead.

The ldap_sasl_bind and ldap_sasl_bind_s APIs can be used to do general and extensible authentication over LDAP through the use of the SASL.

All bind routines take **ld** as their first parameter as returned from ldap_init, ldap_ssl_init or ldap_open.

Simple authentication

The simplest form of the bind call is ldap_simple_bind_s(). It takes the DN to bind as, as well as the user's password (supplied in passwd). It returns an LDAP error indication (see “LDAP_ERROR” on page 50). The ldap_simple_bind() call is

asynchronous, taking the same parameters but only initiating the bind operation and returning the message ID of the request it sent. The result of the operation can be obtained with a subsequent call to ldap_result().

General authentication

The ldap_bind() and ldap_bind_s() routines are deprecated.

They can be used when the authentication method is selected at runtime. They both take an extra method parameter when selecting the authentication method to use. However, when using this toolkit, method must be set to LDAP_AUTH_SIMPLE, to select simple authentication. ldap_bind() and ldap_simple_bind() return the message ID of the initiated request. ldap_bind_s() and ldap_simple_bind_s() return an LDAP error indication on unsuccessful completion, or LDAP_SUCCESS on successful completion.

SASL authentication

Five categories of SASL authentication are supported:

- SASL authentication using the EXTERNAL mechanism
- SASL authentication using the GSSAPI mechanism (Kerberos is supported and implemented as a plug-in)
- SASL authentication using the cram-md5 mechanism (implemented as a plug-in)
- SASL authentication using a user-supplied SASL plug-in library
- SASL authentication using a SASL mechanism implemented by the application itself

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

Also note that the SASL authentication mechanism provides a facility for the LDAP server to return server credentials to the client. An application can obtain the server credentials returned from the server in the SASL bind result with the ldap_parse_sasl_bind_result() API.

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

- ldap_ssl_client_init (initialize the SSL library)
- ldap_ssl_init (host, port, name), where name references a public/private key pair in the client's key database file
- ldap_sasl_bind_s (ld, dn=NULL, mechanism=LDAP_MECHANISM_EXTERNAL, cred=NULL)

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

GSSAPI SASL binds: Kerberos authentication is supported in this release. If the input parameters for ldap_sasl_bind or ldap_sasl_bind_s are mechanism==GSSAPI and cred==NULL, then it is assumed that the user has already authenticated to a Kerberos security server and has obtained a Ticket Granting Ticket (TGT), either through a desktop log-on process, or by using a program such as kinit. The GSSAPI credential handle used to initiate a security context on the LDAP client side is obtained from the current login context. If the input parameters for these two SASL bind functions are mechanism==GSSAPI and cred!=NULL, the caller of

the functions must provide the GSSAPI credential handle for the LDAP client to initiate a security context with an LDAP server. For example, an LDAP server can call a SASL bind function with a credential handle that the server received from a client as a delegated credential handle.

CRAM-MD5 SASL binds: The cram-md5 SASL mechanism is used to hide the credentials on the wire. The cram-md5 plug-in supplied with the IBM Directory Server C-Client SDK implements a multi-bind challenge with the LDAP server. If the multi-bind challenge is successful, the client is authenticated to the server without actually flowing the credentials, for example, a password, in the clear on the wire.

Note: The cram-md5 mechanism is implemented as a SASL bind plug-in. SASL bind plug-ins are only accessible using the synchronous ldap_sasl_bind_s() API. The asynchronous ldap_sasl_bind() API is not supported for use with SASL plug-ins.

See "LDAP_PLUGIN_REGISTRATION" on page 79 for more information about using an LDAP client plug-in. See Chapter 7, "LDAP client plug-in programming reference" on page 169 for more information about developing an LDAP client plug-in.

User-supplied SASL plug-ins: The application developer, or a third party, can implement additional SASL mechanisms, using the IBM Directory Server C-Client's SASL plug-in facility. For example, a client and server SASL plug-in can be developed that supports a new authentication mechanism based upon a retinal scan. If the mechanism associated with this new authentication mechanism is retscan, the application simply invokes ldap_sasl_bind() with mechanism set to retscan. Depending on how the mechanism and plug-in are designed, the application might be required to also supply the user's DN and credentials. Alternatively, the plug-in itself might be responsible for obtaining the user's identity and credentials, which are derived in some way from a retinal scan image.

If the retinal scan plug-in is not defined in ldap.conf, the application must explicitly register the plug-in, using the ldap_register_plugin() API. See "Defining a SASL plug-in" on page 42 for information about defining a SASL plug-in for use with an application. See "LDAP_PLUGIN_REGISTRATION" on page 79 for more information about using an LDAP client plug-in. See Chapter 7, "LDAP client plug-in programming reference" on page 169 for more information about developing an LDAP client plug-in.

SASL mechanisms implemented by the application: In some cases, the SASL mechanism might not require the presence of a plug-in, or any special support in the LDAP library. If the application can invoke the ldap_sasl_bind() or ldap_sasl_bind_s() API with the parameters appropriate to the mechanism, the LDAP library simply encodes the SASL bind request and sends it to the server. If a plug-in is defined for the specified mechanism, the request is diverted to the plug-in, which can perform additional processing before sending the SASL bind to the server.

SASL mechanisms supported by the LDAP server: The application can query the LDAP server's root DSE, using ldap_search() with the following settings:

- base DN set to NULL
- scope set to base
- filter set to "objectclass=*"

If the LDAP server supports one or more SASL mechanisms, the search results include one or more values for the supportedsaslmechanisms attribute type.

Defining a SASL plug-in: When the application issues an ldap_sasl_bind_s() API with a mechanism that is supported by a particular SASL plug-in, the LDAP library must be able to locate the plug-in shared library. Two mechanisms are available for making an LDAP client plug-in known to the LDAP library:

- The plug-in for the specified SASL mechanism is defined in the ldap.conf file. By default, the IBM Directory Server C-Client cram-md5 plug-in is defined in ldap.conf.
- The plug-in has been explicitly registered by the application, using the ldap_register_plugin() API.

See "Finding the Plug-in library" on page 80 for more information about locating a plug-in library and defining plug-ins in the ldap.conf file.

Unbinding

ldap_unbind_ext(), ldap_unbind(), and ldap_unbind_s() are synchronous APIs, in the sense that they send an unbind request to the server, close all open connections associated with the LDAP session handle, and dispose of all resources associated with the session handle before returning. Note that there is no server response to an LDAP unbind operation. All three of the unbind functions return LDAP_SUCCESS or another LDAP error code if the request cannot be sent to the LDAP server. After a call to one of the unbind functions, the session handle ld is invalid and it is illegal to make any further LDAP API calls using the ld.

The ldap_unbind() and ldap_unbind_s() APIs behave identically. The ldap_unbind_ext() API allows server and client controls to be included explicitly, but note that since there is no server response to an unbind request there is no way to receive a response to a server control sent with an unbind request.

Re-binding while following referrals

The ldap_set_rebind_proc() call is used to set the entry-point of a routine that is called back to obtain bind credentials for use when a new server is contacted following an LDAP referral or search reference. Note that this function is only available when LDAP_OPT_REFERRALS is set. This is the default setting. If ldap_set_rebind_proc() is never called, or if it is called with a NULL rebindproc parameter, an unauthenticated simple LDAP bind is always done when chasing referrals. The SSL characteristics of the connections to the referred to servers are preserved when chasing referrals. In addition, if the original bind was an LDAP V3 bind, an LDAP V3 bind is used to connect to the referred-to servers. If the original bind was an LDAP V2 bind, an LDAP V2 bind is used to connect to each referred-to server.

rebindproc must be a function that is declared like the following:

```
int rebindproc( LDAP *ld, char **whop, char **credp,

    int *methodp, int freeit );
```

The LDAP library first calls the rebindproc to obtain the referral bind credentials, and the **freeit** parameter is zero. The **whop**, **credp**, and **methodp** parameters must be set as appropriate. If the rebindproc returns LDAP_SUCCESS, referral processing continues, and the rebindproc is called a second time with freeit non-zero to give your application a chance to free any memory allocated in the previous call.

If anything but LDAP_SUCCESS is returned by the first call to the rebindproc, then referral processing is stopped and that error code is returned for the original LDAP operation.

Errors

Asynchronous routines return -1 in case of error. To obtain the LDAP error, use the ldap_get_errno() API. Synchronous routines return the LDAP error code resulting from the operation.

See also

ldap, ldap_error, ldap_open

LDAP_COMPARE

ldap_compare

ldap_compare_s

ldap_compare_ext

ldap_compare_ext_s

Purpose

Perform an LDAP compare operation.

Synopsis

```
#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,
       const struct berval *bvalue,
       LDAPControl    **serverctrls,
       LDAPControl    **clientctrls,
       int            *msgidp)

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

Input parameters

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

dn Specifies the DN of the entry to perform the compare upon.

attr Specifies the attribute type to use in the comparison.

bvalue
Specifies the attribute value to compare against the entry value. This parameter is used in the ldap_compare_ext and ldap_compare_ext_s routines, and is a pointer to a struct berval, making it possible to compare binary values. See "LDAP_GET_VALUES" on page 57

serverctrls
Specifies a list of LDAP server controls. This parameter can be set to NULL. See "LDAP controls" on page 45 for more information about server controls.

clientctrls
Specifies a list of LDAP client controls. This parameter can be set to NULL. See "LDAP controls" on page 45 for more information about client controls.

Output parameters

msgidp
This result parameter is set to the message ID of the request if the ldap_compare_ext() call succeeds.

Usage

The various LDAP compare routines are used to perform LDAP compare operations. They take **dn**, the DN of the entry upon which to perform the compare, and **attr** and **value**, the attribute type and value to compare to those found in the entry.

The ldap_compare_ext() API initiates an asynchronous compare operation and returns the constant LDAP_SUCCESS if the request was successfully sent, or another LDAP error code if not. If successful, ldap_compare_ext() places the message ID of the request in **msgidp*. A subsequent call to ldap_result() obtains the result of the operation. After the operation has completed, ldap_result() returns the status of the operation in the form of an error code. The error code indicates whether the operation completed successfully (LDAP_COMPARE_TRUE or LDAP_COMPARE_FALSE).

Similarly, the ldap_compare() API initiates an asynchronous compare operation and returns the message ID of that operation. Use a subsequent call to ldap_result(), can be used to obtain the result of the compare. In case of error, ldap_compare() returns -1, setting the session error parameters in the LDAP structure appropriately, which can be obtained by using ldap_get_errno().

See "LDAP_ERROR" on page 50 for more details.

Use the synchronous ldap_compare_s() and ldap_compare_ext_s APIs to perform LDAP compare operations. These APIs return an LDAP error code, which are LDAP_COMPARE_TRUE if the entry contains the attribute value and LDAP_COMPARE_FALSE if it does not. Otherwise, some error code is returned.

The ldap_compare_ext() and ldap_compare_ext_s() APIs both support LDAP V3 server controls and client controls.

Errors

ldap_compare_s() returns an LDAP error code which can be interpreted by calling one of the ldap_error routines. ldap_compare() returns -1 if the initiation request was unsuccessful. It returns the message ID of the request if successful.

See also

ldap, ldap_error

LDAP controls

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

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

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

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

The LDAPControl fields have the following definitions:

ldctl_oid
Specifies the control type, represented as a string.

ldctl_value
Specifies the data associated with the control. Note that the control might not include data.

ldctl_iscritical
Specifies whether the control is critical or not. If the field is non-zero, the operation is carried out only if it is recognized and supported by the server or the client for client-side controls.

LDAP_DELETE

ldap_delete
ldap_delete_s
ldap_delete_ext
ldap_delete_ext_s

Purpose

Perform an LDAP operation to delete a leaf entry.

Synopsis

```
#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)
```

Input parameters

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

dn Specifies the DN of the entry to be deleted.

serverctrls
Specifies a list of LDAP server controls. This parameter can be set to NULL. See "LDAP controls" on page 45 for more information about server controls.

clientctrls
Specifies a list of LDAP client controls. This parameter can be set to NULL. See "LDAP controls" on page 45 for more information about client controls.

Output parameters

msgidp
This result parameter is set to the message ID of the request if the ldap_delete_ext() call succeeds.

Usage

Note: The entry to delete must be a leaf entry, that is, it must have no children. Deletion of entire subtrees in a single operation is not supported by LDAP.

The ldap_delete_ext() API initiates an asynchronous delete operation and returns the constant LDAP_SUCCESS if the request was successfully sent, or another LDAP error code if the request was not successful. If successful, ldap_delete_ext() places the message ID of the request in **msgidp*. ldap_result() returns the status of an operation as an error code. The error code indicates whether the operation completed successfully. ldap_parse_result() checks the error code.

Similarly, the ldap_delete() API initiates an asynchronous delete operation and returns the message ID of that operation. A subsequent call to ldap_result() can be

used to obtain the result of the ldap_delete() operation. In case of error, ldap_delete() returns -1, setting the session error parameters in the LDAP structure appropriately. These error parameters can be obtained by using ldap_get_errno().

See "LDAP_ERROR" on page 50 for more details.

Use the synchronous ldap_delete_s() and ldap_delete_ext_s() APIs to perform LDAP delete operations. The results of both operations are output parameters. These routines return either the constant LDAP_SUCCESS if the operation was successful, or another LDAP error code if the operation was not successful.

Both the ldap_delete_ext() and ldap_delete_ext_s() APIs both support LDAP V3 server controls and client controls.

Errors

ldap_delete_s() returns an LDAP error code which can be interpreted by calling an ldap_error routine. ldap_delete() returns -1 if the request initiation was unsuccessful. It returns the message ID of the request if successful.

See also

ldap, ldap_error

LDAP_FIRST_ENTRY/REFERENCE

ldap_first_entry
ldap_next_entry
ldap_count_entries
ldap_get_entry_controls
ldap_first_reference
ldap_next_reference
ldap_count_references
ldap_parse_reference

Purpose

LDAP result entry and continuation reference parsing and counting routines. Note that APIs with the _np suffix are preliminary implementations, and are not documented in the Internet Draft, "C LDAP Application Program Interface".

Synopsis

```
#include <ldap.h>


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

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

int ldap_count_entries(
                LDAP           *ld,
                LDAPMessage    *result)

int ldap_get_entry_controls_np(
```

```
                LDAP            *ld,
                LDAPMessage     *entry
                LDAPControl     ***serverctrlsp)

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

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

int ldap_count_references(
                LDAP            *ld,
                LDAPMessage     *result)

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

Input parameters

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

result Specifies the result returned by a call to ldap_result() or one of the synchronous search routines, such as ldap_search_s(), ldap_search_st() or ldap_search_ext_s().

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

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

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

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

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

Usage

These routines are used to parse results received from ldap_result() or the synchronous LDAP search operation routines ldap_search_s(), ldap_search_st() and ldap_search_ext_s().

Processing entries

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

The ldap_first_entry() routine is used to retrieve the first entry in a chain of search results. It takes the result returned by a call to ldap_result(), ldap_search_s(), ldap_search_st() or ldap_search_ext_s() and returns a pointer to the first entry in the result.

This pointer must be supplied on a subsequent call to ldap_next_entry() to get the next entry, and so on until ldap_next_entry() returns NULL. ldap_next_entry() returns NULL when there are no more entries. The entries returned from these calls are used in calls to the routines ldap_get_dn(), ldap_first_attribute(), ldap_get_values(), and so forth.

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

Processing continuation references

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

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

The pointer returned from ldap_first_reference() must be supplied on a subsequent call to ldap_next_reference() to get the next continuation reference.

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

Counting entries and references

The ldap_count_entries() API returns the number of entries contained in a search result chain. It can also be used to count the number of entries that remain in a chain if called with a message, entry or continuation reference returned by ldap_first_message(), ldap_next_message(), ldap_first_entry(), ldap_next_entry(), ldap_first_reference() or ldap_next_reference().

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

Errors

If an error occurs in ldap_first_entry(), ldap_next_entry(), ldap_first_reference() or ldap_next_reference(), NULL is returned, and ldap_get_errno() API can be used to obtain the error code.

If an error occurs in ldap_count_entries() or ldap_count_references(), -1 is returned, and ldap_get_errno() can be used to obtain the error code.
ldap_get_entry_controls_np() and ldap_parse_reference_np() return an LDAP error code directly, for example, (LDAP_SUCCESS if the call was successful, an LDAP error if the call was unsuccessful.

See "LDAP_ERROR" for a description of possible error codes.

See also

ldap, ldap_result(), ldap_search(), ldap_first_attribute(), ldap_get_values(), ldap_get_dn()

LDAP_ERROR

ldap_get_errno
ldap_get_lderrno
ldap_set_lderrno
ldap_perror (deprecated)
ldap_result2error (deprecated)
ldap_err2string
ldap_get_exterror

Purpose

LDAP protocol error handling routines.

Synopsis

```
#include <ldap.h>


int ldap_get_errno(
         LDAP        *ld)

int ldap_get_lderrno (
         LDAP        *ld,
         char        **dn,
         char        **errmsg)

int ldap_set_lderrno (
         LDAP        *ld,
         int         errnum,
         char        *dn,
         char        *errmsg)

void ldap_perror(
        LDAP         *ld,
        const char   *s)

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

char *ldap_err2string(
        int          error)

int ldap_get_exterror(
         LDAP        *ld)
```

Input parameters

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

dn Specifies a DN that identifies an existing entry, indicating how much of the name in the request was recognized by the server. The DN is returned when an LDAP_NO_SUCH_OBJECT error is returned from the server. The matched DN string must be freed by calling ldap_memfree().

errmsg
The text of the error message, as returned from the server. The error message string must be freed by calling ldap_memfree().

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

res Specifies the result, as produced by ldap_result() or ldap_search_s(), to be converted to the error code with which it is associated.

freeit Specifies whether or not the result, **res**, must be freed as a result of calling ldap_result2error(). If non-zero, the result, **res**, is freed by the call. If zero, **res** is not freed by the call.

errnum
The LDAP error code, as returned by ldap_parse_result() or another LDAP API call.

Usage

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

The ldap_get_errno() and ldap_get_lderrno() APIs obtain information for the most recent error that occurred for an LDAP operation. When an error occurs at the LDAP server, the server returns the following information back to the client:

- The LDAP result code for the error that occurred.
- A message containing any additional information about the error from the server.

If the error occurred because an entry specified by a DN cannot be found, the server might also return the portion of the DN that identifies an existing entry.

Both APIs return the server's error result code. Use ldap_get_lderrno() to obtain the message and matched DN.

The ldap_set_lderrno() API sets an error code and other information about an error in the specified LDAP structure. This function can be called to set error information that is retrieved by subsequent ldap_get_lderrno() calls.

The ldap_result2error() routine takes **res**, a result as produced by ldap_result or ldap_search_s, and returns the corresponding error code. Possible error codes follow (see "Errors" on page 52). If the freeit parameter is non-zero, it indicates that the res parameter must be freed by a call to ldap_msgfree() after the error code has been extracted. The ld_errno field in ld is set and returned.

The returned value can be passed to ldap_err2string(), which returns a pointer to a character string which is a textual description of the LDAP error code. The character string must not be freed when use of the string is complete.

The ldap_perror() routine can be called to print an indication of the error on standard error.

The ldap_get_exterror() routine returns the current extended error code returned by an LDAP server or other library, such as Kerberos or SSL, for the LDAP session. For some error codes, it might be possible to further interpret the error condition. For example, for SSL errors the extended error code might indicate why an SSL handshake failed.

Errors

The possible values for an LDAP error code are:

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

#define LDAP_REFERRAL                       0X0a
#define LDAP_ADMIN_LIMIT_EXCEEDED           0X0b
#define LDAP_UNAVAILABLE_CRITICAL_EXTENSION 0X0c
#define LDAP_CONFIDENTIALITY_REQUIRED       0x0d
#define LDAP_SASLBIND_IN_PROGRESS           0x0e

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

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

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

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

#define LDAP_AFFECTS_MULTIPLE_DSAS          0X47
```

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

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

#define LDAP_CLIENT_LOOP                    0x64
#define LDAP_REFERRAL_LIMIT_EXCEEDED        0x65

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

#define LDAP_NO_EXPLICIT_OWNER              0x80
#define LDAP_NO_LOCK                        0x81

/* DNS related error codes */
#define LDAP_DNS_NO_SERVERS                 0x85
        /* No LDAP servers found */
#define LDAP_DNS_TRUNCATED                  0x86
        /* Warning: truncated DNS results */
#define LDAP_DNS_INVALID_DATA               0x87
        /* Invalid DNS Data */
#define LDAP_DNS_RESOLVE_ERROR              0x88
        /* Can't resolve system domain or nameserver */
#define LDAP_DNS_CONF_FILE_ERROR            0x89
        /* DNS Configuration file error */

/* UTF8 related error codes */
#define LDAP_XLATE_E2BIG                    0xA0
         /* Output buffer overflow */
#define LDAP_XLATE_EINVAL                   0xA1
         /* Input buffer truncated */
#define LDAP_XLATE_EILSEQ                   0xA2
         /* Unusable input character */
#define LDAP_XLATE_NO_ENTRY                 0xA3
         /* No codeset point to map to */
```

See also

ldap, ldap_memfree, ldap_parse routines

LDAP_EXTENDED_OPERATION

ldap_extended_operation

ldap_extended_operation_s

Purpose

Perform extended operations and parse extended result.

Synopsis

```
#include <ldap.h>


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

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

Input parameters

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

reqoid Specifies the dotted-object identifier (OID) text string that identifies the extended operation to be performed by the server.

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

serverctrls
Specifies a list of LDAP server controls. This parameter can be set to NULL. See "LDAP controls" on page 45 for more information about server controls.

clientctrls
Specifies a list of LDAP client controls. This parameter can be set to NULL. See "LDAP controls" on page 45 for more information about client controls.

Output parameters

msgidp
This result parameter is set to the message ID of the request if the ldap_extended_operation() call is successfully sent to the server. To check the result of this operation, call the ldap_result() and ldap_parse_result() APIs. The server can also return an OID and result data. Because the asynchronous ldap_extended_operation does not directly return the results, use ldap_parse_extended_result() to get the results.

retoidp
This result parameter is set to point to a character string that is set to an allocated, dotted-OID text string returned from the server. This string must be disposed of using the ldap_memfree() API. If no OID is returned, *retoidp is set to NULL.

retdatap
This result parameter is set to a pointer to a berval structure pointer that is set to an allocated copy of the data returned by the server. This struct berval must be disposed of using ber_bvfree(). If no data is returned, *retdatap is set to NULL

Usage

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

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

If the LDAP server does not support the extended operation, the server rejects the request. IBM Directory Server version 4.1 provides a server plug-in interface that can be used to add extended operation support. For more information, see the *IBM Directory Server Version 4.1: Server Plug-ins Reference*.

To determine if the requisite extended operation is supported by the server, get the rootDSE of the LDAP server, and check for the supportedExtension attribute. If the values for this attribute include the OID of your extended operation, then the server supports the extended operation. If the supportedExtension attribute is not present in the rootDSE, then the server is not configured to support any extended operations.

Errors

The ldap_extended_operation_s API returns the LDAP error code for the operation.

ldap_extended_operation() returns -1 instead of a valid msgid if an error occurs, setting the session error in the LD structure, which can be obtain by using ldap_get_errno().

See “LDAP_ERROR” on page 50 for more details.

Notes

These routines allocate storage. Use ldap_memfree to free the returned OID. Use ber_bvfree to free the returned struct berval.

See also

ldap, ldap_result, ldap_error

LDAP_GET_DN

ldap_dn2ufn
ldap_get_dn
ldap_explode_dn
ldap_explode_dns
ldap_explode_rdn

Purpose

LDAP DN and RDN handling routines.

Synopsis

```
#include <ldap.h>


char *ldap_dn2ufn(
        const char *dn)

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

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

char **ldap_explode_dns(
         const char *dn)

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

Input parameters

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

dn Specifies the DN to be exploded (as returned from ldap_get_dn()), or converted to a friendlier form (as returned from ldap_dn2ufn()).

rdn Specifies the RDN to be exploded (as returned from ldap_explode_dn()).

entry The entry whose dn is to be retrieved.

notypes
Specifies if type names are to be returned for each RDN. If non-zero, the type information is stripped. If zero, the type information is retained. For example, setting notypes to 1 can result in the RDN ″cn=Fido″ being returned as Fido.

Usage

The ldap_dn2ufn() routine takes a DN and converts it into a friendlier representation by removing the attribute type that is associated with each RDN. For example, the DN ″cn=John Doe, ou=Widget Division, ou=Austin, o=IBM, c=US″ is returned in its friendly form as ″John Doe, Widget Division, Austin, IBM, US″. Space for the user-friendly name is obtained by the LDAP API, and must be freed by a call to ldap_memfree().

The ldap_get_dn() routine takes an entry as returned by ldap_first_entry() or ldap_next_entry() and returns a copy of the entry's DN. Space for the DN is obtained by the LDAP API, and must be freed by a call to ldap_memfree().

The ldap_explode_dn() routine takes a DN (perhaps as returned by ldap_get_dn()) and breaks it up into its component parts. Each part is known as a Relative Distinguished Name, or RDN. ldap_explode_dn() returns a NULL-terminated array of character strings, each component of which contains an RDN from the DN. The notypes parameter is used to request that only the RDN values be returned, not their types. For example, the DN "cn=Bob,c=US" returns an array as either {"cn=Bob","c=US",NULL} or {"Bob","US",NULL} depending on whether notypes was 0 or 1. The result can be freed by calling ldap_value_free().

The ldap_explode_dns() routine takes a DNS-style DN and breaks it up into its component parts. It returns a NULL-terminated array of character strings. For example, the DN "austin.ibm.com" returns { "austin", "ibm", "com", NULL }. The result can be freed by calling ldap_value_free().

The ldap_explode_rdn() routine takes an RDN (perhaps as returned by ldap_explode_dn()) and breaks it up into its component parts. ldap_explode_rdn() returns a NULL-terminated array of character strings. The notypes parameter is used to request that only the component values be returned, not their types. For example, the RDN "ou=Research + cn=Bob" returns as either {"ou=Research", "cn=Bob", NULL} or {"Research","Bob", NULL}, depending on whether notypes was 0 or 1. The result can be freed by calling ldap_value_free().

Errors

If an error occurs in ldap_dn2ufn(), ldap_get_dn(), ldap_explode_dn() or ldap_explode_rdn(), NULL is returned. If ldap_get_dn() returns NULL, the ldap_get_errno() API can be used to obtain the error code. See "LDAP_ERROR" on page 50 for a description of possible error codes.

Notes

These routines allocate memory that the caller must deallocate.

See also

ldap, ldap_first_entry, ldap_error, ldap_value_free

LDAP_GET_VALUES

ldap_get_values
ldap_get_values_len
ldap_count_values
ldap_count_values_len
ldap_value_free
ldap_value_free_len

Purpose

LDAP attribute value handling routines.

Synopsis

```
#include <ldap.h>


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


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

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

int ldap_count_values(
          char          **vals)

int ldap_count_values_len(
          struct berval **bvals)

void ldap_value_free(
          char          **vals)

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

Input parameters

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

attr Specifies the attribute whose values are desired.

entry Specifies an LDAP entry as returned from ldap_first_entry() or ldap_next_entry().

vals Specifies a pointer to a NULL-terminated array of attribute values, as returned by ldap_get_values().

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

Usage

These routines are used to retrieve and manipulate attribute values from an LDAP entry as returned by ldap_first_entry() or ldap_next_entry().

An attribute's values can be represented in two forms:

- A NULL-terminated array of strings. This representation is appropriate when the attribute contains string data, for example, a title, description or name.
- A NULL-terminated array of berval structures. This representation is appropriate when the attribute contains binary data, for example, a JPEG file.

String values

Use ldap_get_values() to obtain attribute values as an array of strings. The ldap_get_values() API takes the entry and the attribute attr whose values are desired and returns a NULL-terminated array of character strings which represent

the attribute's values. attr can be an attribute type as returned from ldap_first_attribute() or ldap_next_attribute() or if the attribute type is known it can simply be provided.

The number of values in the array of character strings can be counted by calling ldap_count_values(). The array of values returned can be freed by calling ldap_value_free().

If your application is designed to rely on the LDAP library to convert LDAP V3 string data from UTF-8 to the local code page (enabled on a per-connection basis by using the ldap_set_option() API with the LDAP_OPT_UTF8_IO), strings returned in the NULL-terminated array of string values can contain multi-byte characters, as defined in the local code page. In this case, the application must use string handling routines that are properly enabled to handle multi-byte strings.

Binary values

If the attribute values are binary in nature, and thus not suitable to be returned as an array of character strings, the ldap_get_values_len() routine can be used instead. It takes the same parameters as ldap_get_values(), but returns a NULL-terminated array of pointers to berval structures, each containing the length of, and a pointer to, a value.

The number of values in the array of bervals can be counted by calling ldap_count_values_len(). The array of values returned can be freed by calling ldap_value_free_len().

Errors

If an error occurs in ldap_get_values() or ldap_get_values_len(), NULL is returned and the ldap_get_errno() API can be used to obtain the error code. See "LDAP_ERROR" on page 50 for a description of possible error codes.

See also

ldap, ldap_first_entry, ldap_first_attribute, ldap_error

LDAP_INIT

ldap_init
ldap_open (deprecated)
ldap_set_option
ldap_get_option
ldap_version

Purpose

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

Synopsis

```
#include <ldap.h>


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

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

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

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

int ldap_version(
         LDAPVersion  *version)
```

Input parameters

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

host Several methods are supported for specifying one or more target LDAP servers, including the following:

Explicit Host List

Specifies the name of the host on which the LDAP server is running. The host parameter can contain a blank-separated list of hosts to try to connect to, and each host can optionally be of the form host:port. If present, the :port overrides the port parameter supplied on ldap_init(), ldap_ssl_init() or ldap_open(). The following are typical examples:

```
ld=ldap_init ("server1", ldap_port);
ld=ldap_init ("server2:1200", ldap_port);
ld=ldap_init ( "server1:800 server2:2000 server3", ldap_port);
```

Localhost

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

Default Hosts

If the host parameter is set to ″ldap://″ the LDAP library attempts to locate one or more default LDAP servers, with non-SSL ports, using the IBM Directory Server ldap_server_locate() function. The port specified on the call is ignored, since ldap_server_locate() returns the port. For example, the following two are equivalent:

```
ld=ldap_init ("ldap://", ldap_port);
d=ldap_init (LDAP_URL_PREFIX, LDAP_PORT);
```

If more than one default server is located, the list is processed in sequence, until an active server is found.

The LDAP URL can include a Distinguished Name, used as a filter for selecting candidate LDAP servers based on the server's suffixes. If the most significant portion of the DN is an exact match with a server's suffix after normalizing for case, the server is added to the list of candidate servers. For example, the following returns default LDAP servers that have a suffix that supports the specified DN only:

```
ld=ldap_init ("ldap:///cn=fred, dc=austin,
        dc=ibm, dc=com", LDAP_PORT);
```

In this case, a server that has a suffix of ″dc=austin, dc=ibm, dc=com″ matches. If more than one default server is located, the list is processed in sequence, until an active server is found.

If the LDAP URL contains a host name and optional port, the host is used to create the connection. No attempt is made to locate the default servers, and the DN, if present, is ignored. For example, the following two are equivalent:

```
ld=ldap_init ("ldap://myserver", LDAP_PORT);
ld=ldap_init ("myserver", LDAP_PORT);
```

See “Locating default LDAP servers” on page 69 for more information about the algorithm used to locate default LDAP servers.

Local Socket

If the host parameter is prefixed with a forward slash (/), the host parameter is assumed to be the name of a Unix socket, that is, family is AF_UNIX, and port is ignored. Use of a Unix socket requires the LDAP server to be running on the local host. In addition, the local operating system must support Unix sockets and the LDAP server must be listening on the specified Unix socket. Unix variants of the IBM Directory Server listen on the /tmp/s.slapd local socket, in addition to any configured TCP/IP ports. For example:

```
ld=ldap_init ("/tmp/s.slapd", ldap_port);
```

Host with Privileged Port

On platforms that support the rresvport function, typically Unix platforms, if a specified host is prefixed with ″privport://″, then the LDAP library uses the rresvport() function to attempt to obtain one of the reserved ports (512 through 1023), instead of an ephemeral port. The search for a reserved port starts at 1023 and stops at 512. If a reserved port cannot be obtained, the function call fails. For example:

```
ld=ldap_init ("privport://server1", ldap_port);
ld=ldap_init ("privport://server2:1200", ldap_port);
ld=ldap_init ("privport://server1:800 server2:2000
  privport://server3", ldap_port);
```

port Specifies the port number to connect to. If the default IANA-assigned port of 389 is desired, LDAP_PORT must be specified. To use the default SSL port 636 for SSL connections, use LDAPS_PORT.

optionToSet

Identifies the option value that is to be set on the ldap_set_option() call. See “Usage” on page 62 for the list of supported options.

optionToGet

Identifies the option value that is to be queried on the ldap_get_option() call. See “Usage” on page 62 for the list of supported options.

optionValue

Specifies the address of the value to set using ldap_set_option() or the address of the storage in which the queried value is returned using ldap_get_option().

version
Specifies the address of an LDAPVersion structure that contains the following returned values:

sdk_version
SDK version, multiplied by 100.

protocol_version
Highest LDAP protocol supported, multiplied by 100.

SSL_version
SSL version supported, multiplied by 100.

security_level
Level of encryption supported, in bits. Set to LDAP_SECURITY_NONE if SSL not enabled.

ssl_max_cipher
A string containing the default ordered set of ciphers supported by this installation. See "LDAP_SET_OPTION syntax for LDAP V2 applications" on page 69 for more information about changing the set of ciphers used to negotiate the secure connection with the server.

sdk_vendor
A pointer to a static string that identifies the supplier of the LDAP library. This string must not be freed by the application.

sdk_build_level
A pointer to a static string that identifies the build level, including the date when the library was built. This string must not be freed by the application.

Usage

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

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

The ldap_init() and ldap_open() APIs return a pointer to an LDAP structure, which must be passed to subsequent calls to ldap_set_option(), ldap_simple_bind(), ldap_search(), and so forth.

The LDAP structure is opaque to the application. Direct manipulation of the LDAP structure is not recommended. The ldap_version() API returns the toolkit version (multiplied by 100). It also sets information in the LDAPVersion structure (see 61).

Setting and getting session settings

The ldap_set_option() API sets options for the specified LDAP connection. The ldap_get_option() API queries settings associated with the specified LDAP connection.

The following session settings can be set and retrieved using the ldap_set_option and ldap_get_option API:

LDAP_OPT_SIZELIMIT
Get/Set maximum number of entries that can be returned on a search operation.

LDAP_OPT_TIMELIMIT
Get/Set maximum number of seconds to wait for search results.

LDAP_OPT_REFHOPLIMIT
Get/Set maximum number of referrals in a sequence that the client can follow.

LDAP_OPT_DEREF
Get/Set rules for following aliases at the server.

LDAP_OPT_REFERRALS
Get/Set whether or not referrals must be followed by the client.

LDAP_OPT_DEBUG
Get/Set debug options.

LDAP_OPT_SSL_CIPHER
Get/Set SSL ciphers to use.

LDAP_OPT_SSL_TIMEOUT
Get/Set SSL timeout for refreshing session keys.

LDAP_OPT_REBIND_FN
Get/Set address of application's setrebindproc procedure.

LDAP_OPT_PROTOCOL_VERSION
Get/Set LDAP protocol version to use (V2 or V3).

LDAP_OPT_SERVER_CONTROLS
Get/Set default server controls.

LDAP_OPT_CLIENT_CONTROLS
Get/Set default client library controls.

LDAP_OPT_UTF8_IO
Get/Set mode for converting string data between the local code page and UTF-8.

LDAP_OPT_HOST_NAME
Get current host name (cannot be set).

LDAP_OPT_ERROR_NUMBER
Get error number (cannot be set).

LDAP_OPT_ERROR_STRING
Get error string (cannot be set).

LDAP_OPT_API_INFO
Get API version information (cannot be set).

LDAP_OPT_EXT_ERROR
Get extended error code.

If your LDAP application is based on the LDAP V2 APIs and uses the ldap_set_option() or ldap_get_option() functions, that is, you are using ldap_open, or your application uses ldap_init and ldap_set_option to switch from the default of LDAP V3 to use the LDAP V2 protocol and subsequently uses the ldap_set_option() or ldap_get_option() calls, see "LDAP_SET_OPTION syntax for LDAP V2 applications" on page 69 for important information.

Additional details on specific options for ldap_set_option() and ldap_get_option are provided in the following sections.

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

Note: The actual size limit for operations is also bounded by the maximum number of entries that the server is configured to return. Thus, the actual size limit is the lesser of the value specified on this option and the value configured in the LDAP server. The default sizelimit is unlimited, specified with a value of zero, thus deferring to the sizelimit setting of the LDAP server.

For example:

```
sizevalue=50;
ldap_set_option( ld, LDAP_OPT_SIZELIMIT, &sizevalue);
ldap_get_option( ld, LDAP_OPT_SIZELIMIT, &sizevalue);
```

LDAP_OPT_TIMELIMIT: Specifies the number of seconds to wait for search results.

Note: The actual time limit for operations is also bounded by the maximum time that the server is configured to allow. Thus, the actual time limit is the lesser of the value specified on this option and the value configured in the LDAP server.

The default is unlimited (specified with a value of zero). For example:

```
timevalue=50;
ldap_set_option( ld, LDAP_OPT_TIMELIMIT, &timevalue);
ldap_get_option( ld, LDAP_OPT_TIMELIMIT, &timevalue);
```

LDAP_OPT_REFHOPLIMIT: Specifies the maximum number of hops that the client library takes when chasing referrals. The default is 10. For example:

```
hoplimit=7;
ldap_set_option( ld, LDAP_OPT_REFHOPLIMIT, &hoplimit);
ldap_get_option( ld, LDAP_OPT_REFHOPLIMIT, &hoplimit);
```

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

Supported values:

LDAP_DEREF_NEVER 0

LDAP_DEREF_SEARCHING 1

LDAP_DEREF_FINDING 2

LDAP_DEREF_ALWAYS 3

For example:

```
int deref = LDAP_DEREF_NEVER;
ldap_set_option( ld, LDAP_OPT_DEREF, &deref);
ldap_get_option( ld, LDAP_OPT_DEREF, &deref);
```

LDAP_OPT_REFERRALS: Specifies whether the LDAP library automatically follows referrals returned by LDAP servers or not. It can be set to one of the constants LDAP_OPT_ON or LDAP_OPT_OFF. By default, the LDAP client follows referrals. For example:

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

LDAP_OPT_DEBUG: Specifies a bit-map that indicates the level of debug trace for the LDAP library.

Supported values:

```
/* Debug levels */

LDAP_DEBUG_OFF         0x000
LDAP_DEBUG_TRACE       0x001
LDAP_DEBUG_PACKETS     0x002
LDAP_DEBUG_ARGS        0x004
LDAP_DEBUG_CONNS       0x008
LDAP_DEBUG_BER         0x010
LDAP_DEBUG_FILTER      0x020
LDAP_DEBUG_CONFIG      0x040
LDAP_DEBUG_ACL         0x080
LDAP_DEBUG_STATS       0x100
LDAP_DEBUG_STATS2      0x200
LDAP_DEBUG_SHELL       0x400
LDAP_DEBUG_PARSE       0x800
LDAP_DEBUG_ANY         0xffff
```

For example:

```
int value;
 int debugvalue= LDAP_DEBUG_TRACE | LDAP_DEBUG_PACKETS;
 ldap_set_option( ld, LDAP_OPT_DEBUG, &debugvalue);
 ldap_get_option( ld, LDAP_OPT_DEBUG, &value );
```

LDAP_OPT_SSL_CIPHER: Specifies a set of one or more ciphers to be used when negotiating the cipher algorithm with the LDAP server. Choose the first cipher in the list that is common with the list of ciphers supported by the server. For the export version of the library, the value used is ″090306″. For the domestic version of the library, the default value is ″05040A090306″.

Supported ciphers:

LDAP_SSL_RC4_MD5_EX ″03″

LDAP_SSL_RC2_MD5_EX ″06″

LDAP_SSL_RC4_SHA_US ″05″ (Non-export only)

LDAP_SSL_RC4_MD5_US ″04″ (Non-export only)

LDAP_SSL_DES_SHA_US ″09″

LDAP_SSL_3DES_SHA_US ″0A″ (Non-export only)

For example:

```
char *setcipher = "090A";
char *getcipher;
ldap_set_option( ld, LDAP_OPT_SSL_CIPHER, setcipher);
ldap_get_option( ld, LDAP_OPT_SSL_CIPHER, &getcipher );
```

Use ldap_memfree() to free the memory returned by the call to ldap_get_option().

LDAP_OPT_SSL_TIMEOUT: Specifies in seconds the SSL inactivity timer. After the specified seconds, in which no SSL activity has occurred, the SSL connection is refreshed with new session keys. A smaller value can help increase security, but has a small impact on performance. The default SSL timeout value is 43200 seconds. For example:

```
value = 100;
ldap_set_option( ld, LDAP_OPT_SSL_TIMEOUT, &value );
ldap_get_option( ld, LDAP_OPT_SSL_TIMEOUT, &value)
```

LDAP_OPT_REBIND_FN: Specifies the address of a routine to be called by the LDAP library to authenticate a connection with another LDAP server when chasing a referral or search reference. If a routine is not defined, referrals are chased using the identity and credentials specified on the bind sent to the original server. A default routine is not defined. For example:

```
extern LDAPRebindProc proc_address;
LDAPRebindProc value;
ldap_set_option( ld, LDAP_OPT_REBIND_FN, &proc_address);
ldap_get_option( ld, LDAP_OPT_REBIND_FN, &value);
```

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

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

LDAP_OPT_SERVER_CONTROLS: Specifies a default list of server controls to be sent with each request. The default list can be overridden by specifying a server control, or list of server controls, on specific APIs. By default, there are no settings for Server Controls. For example:

```
ldap_set_option( ld, LDAP_OPT_SERVER_CONTROLS, &ctrlp);
```

LDAP_OPT_CLIENT_CONTROLS: Specifies a default list of client controls to be processed by the client library with each request. Since client controls are not defined for this version of the library, the ldap_set_option() API can be used to define a set of default, non-critical client controls. If one or more client controls in the set is critical, the entire list is rejected with a return code of

```
LDAP_UNAVAILABLE_CRITICAL_EXTENSION
```

LDAP_OPT_UTF8_IO: Specifies whether the LDAP library automatically converts string data to and from the local code page. It can be set to either LDAP_UTF8_XLATE_ON or LDAP_UTF8_XLATE_OFF. By default, the LDAP library does not convert string data.

When conversion is disabled by default, the LDAP library assumes that data received from the application using LDAP APIs is already represented in UTF-8. Similarly, the LDAP library assumes that the application is prepared to receive string data from the LDAP library represented in UTF-8, or as binary.

When LDAP_UTF8_XLATE_ON is set, the LDAP library assumes that string data received from the application using LDAP APIs is in the default (or explicitly designated) code page. Similarly, all string data returned from the LDAP library back to the application is converted to the designated local code page.

It is important to note that only string data supplied on connection-based APIs is translated, that is, only those APIs that include an ld are subject to translation.

It is also important to note that translation of strings from a UTF-8 encoding to local code page can result in loss of data when one or more characters in the UTF-8 encoding cannot be represented in the local code page. When this occurs, a substitution character replaces any UTF-8 characters that cannot be converted to the local code page.

For more information on explicitly setting the locale for conversions, see ldap_set_locale(). For example:

```
int value;
ldap_set_option( ld, LDAP_OPT_UTF8_IO, (void*)LDAP_UTF8_XLATE_ON);
 ldap_get_option( ld, LDAP_OPT_UTF8_IO, &value);
```

LDAP_OPT_HOST_NAME: This is a read-only option that returns a pointer to the hostname for the original connection (as specified on ldap_init(), ldap_open(), or ldap_ssl_init()). For example:

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

Use ldap_memfree to free the memory returned by the call to ldap_get_option().

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

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

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

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

Use ldap_memfree() to free the memory returned by the call to ldap_get_option().

LDAP_OPT_API_INFO: This is a read-only option that returns basic information about the API and about the specific implementation being used. The ld parameter to ldap_get_option() can be either NULL or a valid LDAP session handle which was obtained by calling ldap_init(), ldap_ssl_init() or ldap_open(). The optdata parameter to ldap_get_option() must be the address of an LDAPAPIInfo structure which is defined as follows:

```
typedef struct ldapapiinfo {
    int  ldapai_info_version;     /* version of this struct (1) */
    int  ldapai_api_version;      /* revision of API supported */
    int  ldapai_protocol_version; /* highest LDAP version supported */
    char **ldapai_extensions;     /* names of API extensions */
    char *ldapai_vendor_name;     /* name of supplier */
    int  ldapai_vendor_version;   /* supplier-specific version times 100 */
} LDAPAPIInfo;
```

Note: The ldapai_info_version field of the LDAPAPIInfo structure must be set to the value LDAP_API_INFO_VERSION before calling ldap_get_option() so that it can be checked for consistency. All other fields are set by the ldap_get_option() function.

The members of the LDAPAPIInfo structure are:

ldapai_info_version
A number that identifies the version of the LDAPAPIInfo structure. This must be set to the value LDAP_API_INFO_VERSION before calling ldap_get_option(). If the value received is not recognized by the API implementation, the ldap_get_option() function sets ldapai_info_version to a valid value that can be recognized, sets ldapai_api_version to the correct value, and returns an error without filling in any of the other fields in the LDAPAPIInfo structure.

ldapai_api_version
A number that matches that assigned to the C LDAP API RFC supported by the API implementation. This number must match the value of the LDAP_API_VERSION define.

ldapai_protocol_version
The highest LDAP protocol version supported by the implementation. For example, if LDAPv3 is the highest version supported then this field is set to 3.

ldapai_extensions
A NULL-terminated array of character strings that lists the names of API extensions. The caller is responsible for disposing of the memory occupied by this array by passing it to ldap_value_free().

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

```
int error;
ldap_get_option( ld, LDAP_OPT_EXT_ERROR, &exterror);
```

LDAP_OPT_EXT_ERROR returns errors reported by the SSL library.

Errors

If an error occurs, a non-zero return code is returned from ldap_set_option and ldap_get_option.

LDAP_DEBUG

To obtain debug information from a client application built using the IBM Directory Server LDAP C-API, you can set the environment variables LDAP_DEBUG and LDAP_DEBUG_FILE.

For Unix, enter the following command before running your application:

```
export LDAP_DEBUG=65535
```

For the Windows NT and Windows 2000 operating systems, enter the following command before running your application:

```
set LDAP_DEBUG=65535
```

Trace messages in the LDAP C-API library are output to standard error. Use LDAP_DEBUG_FILE=*xxxxx* to send the trace output to the file *xxxxx*.

These environment variables affect only applications run in the same shell (or command window) session. You can also call ldap_set_option() in your application to enable and disable the library's trace messages.

LDAP_SET_OPTION syntax for LDAP V2 applications

To maintain compatibility with older versions of the LDAP client library (pre-LDAP V3), the ldap_set_option() API expects the value of the following option values to be supplied, instead of the address of the value, when the application is running as an LDAP V2 application:

- LDAP_OPT_SIZELIMIT
- LDAP_OPT_TIMELIMIT
- LDAP_OPT_SSL_TIMEOUT
- LDAP_OPT_DEREF
- LDAP_OPT_DEBUG

The value returned by ldap_get_option() when LDAP_OPT_PROTOCOL_VERSION is specified can be used to determine how parameters must be passed to the ldap_set_option() call. The easiest way to work with this compatibility feature is to guarantee that calls to ldap_set_option() are all performed while the LDAP_OPT_PROTOCOL_VERSION is set to the same value. If this cannot be guaranteed by the application, then follow the format of the example below when coding the call to ldap_set_option():

```
int sizeLimit=100;

int protocolVersion;

ldap_get_option( ld, LDAP_OPT_PROTOCOL_VERSION, &protocolVersion );

if ( protocolVersion == LDAP_VERSION2 ) {
   ldap_set_option( ld, LDAP_OPT_SIZELIMIT, (void *)sizeLimit );
} else { /* the protocol version is LDAP_VERSION3 */
   ldap_set_option( ld, LDAP_OPT_SIZELIMIT, &sizeLimit );
}
```

The LDAP application is typically running as LDAP V2 when it uses ldap_open() to create the LDAP connection. The LDAP application is typically running as LDAP V3 when it uses ldap_init() to create the LDAP connection. However, it was possible with the LDAP V2 API to call ldap_init(), so there can be cases where this is not true. Note that LDAP_OPT_PROTOCOL_VERSION can be used to toggle the protocol, in which case the behavior of ldap_set_option() changes.

Locating default LDAP servers

When the ldap_init, ldap_open or ldap_ssl_init APIs are invoked with an LDAP URL of the following forms, the ldap_server_locate() function is used to obtain a set of one or more default LDAP servers:

```
ld=ldap_init ("ldap://", ldap_port);          /* locate servers with
        non-secure ports */
ld=ldap_ssl_init ("ldaps://", ldap_port);     /* locate servers with
        secure SSL ports */
```

The ldap_server_locate() API provides several options for searching for default LDAP servers. An application using ldap_server_locate() in an explicit fashion can control these options. When ldap_server_locate() is used implicitly, as described here, the following options are used:

Security

If the non-secure LDAP URL is specified (ldap://), servers with a non-secure security type are used as candidate servers only. If the secure LDAP URL is specified, (ldaps://), servers with a Secure security type are used as candidate servers only.

Source for Server Information
The ldap_server_locate() API can be used to find default LDAP server information in either a local configuration file, or published in the Domain Name System (DNS). In this case, the default behavior is used. The ldap_server_locate() API looks for a local configuration file first, and attempts to find one or more LDAP servers that meet the search criteria (security and suffix filter). If nothing is found, it then searches DNS. See ldap_server_conf_save() for additional information about using a local configuration file.

DNS Domain Name
When searching the local configuration and DNS, the ldap_server_locate() API assumes that your default LDAP servers are published in your locally configured TCP/DNS domain name space, for example, acme.com.

Service Name and Protocol
A complete search is performed using ldap for the service name and tcp for the protocol. If no servers are located, the search is rerun, using _ldap and _tcp.

Note: If the default behavior as described here is not appropriate for your application, consider using the ldap_server_locate() API explicitly, prior to invoking the ldap_init() or ldap_ssl_init() API.

Multithreaded applications

The LDAP client libraries are generally thread safe. While a multithreaded application can safely use the LDAP library on multiple threads within the application, there are a few considerations to keep in mind:

- Using the LDAP connection, that is, the ld, on the thread that is created is a good model. This avoids the possibility of conflicts which can arise if multiple threads are concurrently processing the results of an operation submitted on a different thread.
- An application can be designed to submit requests on one or more threads, with results being fetched on different threads. This is also a good model, since it avoids the situation where two threads are attempting to process the results associated with a single LDAP connection.
- The ldap_get_errno() API obtains information with respect to the most recent error that occurred for the specified LDAP connection. It does not return the most recent LDAP error that occurred on the thread on which it is issued.
- A key consideration is that only a single thread must be performing operations on a particular LDAP connection at any one point in time.
- Note that the locale is applicable to all conversions by the LDAP library within the applications address space. The LDAP locale must be set or changed only when there is no other LDAP activity occurring within the application on other threads.

Notes

Do not make any assumptions about the order or location of elements in the opaque LDAP structure.

See also

ldap, ldap_bind

LDAP_MEMFREE

ldap_memfree
ldap_ber_free
ldap_control_free
ldap_controls_free
ldap_msgfree

Purpose

Free storage allocated by the LDAP library.

Synopsis

```
#include <ldap.h>


void ldap_memfree(
        char             *mem)

void ldap_ber_free(
        BerElement       *berptr)

void ldap_control_free (
        LDAPControl      *ctrl)

void ldap_controls_free)
        LDAPControl      **ctrls)

int ldap_msgfree(
       LDAPMessage       *msg)
```

Input parameters

mem Specifies the address of storage that was allocated by the LDAP library.

berptr Specifies the address of the BerElement returned from ldap_first_attribute() and ldap_next_attribute().

ctrl Specifies the address of an LDAPControl structure.

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

Usage

ldap_memfree() is used to free storage that has been allocated by the LDAP library (libldap). Use this routine as directed when using ldap_error(), ldap_get_option(), ldap_first_attribute(), ldap_default_dn_get() and ldap_enetwork_domain_get().

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

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

The ldap_msgfree() routine is used to free the memory allocated for an LDAP message by ldap_result, ldap_search_s, ldap_search_ext_s() or ldap_search_st(). It takes a pointer to the result to be freed and returns the type of the message it freed.

The ldap_ber_free() routine is used to free the BerElement pointed to by berptr. The LDAP library automatically frees the BerElement when ldap_next_attribute() returns NULL. The application is responsible for freeing the BerElement if it does not invoke ldap_next_attribute() until it returns NULL.

See also

ldap, ldap_controls

LDAP_MESSAGE

ldap_first_message

ldap_next_message

ldap_count_messages

Purpose

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

Synopsis

```
#include <ldap.h>


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

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

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

Input parameters

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

result Specifies the result returned by a call to ldap_result() or one of the synchronous search routines (ldap_search_s(), ldap_search_st() or ldap_search_ext_s()).

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

Usage

These routines are used to step through the list of messages in a result chain, as returned by ldap_result().

For search operations, the result chain can include:

- Referral messages
- Entry messages
- Result messages

The ldap_count_messages() API is used to count the number of messages returned. The ldap_msgtype() API can be used to distinguish between the different message types. Unlike ldap_first_entry(), ldap_first_message() returns either of the three types of messages.

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

The ldap_count_messages API can also be used to count the number of messages that remain in a chain if called with a message, entry, or reference returned by ldap_first_message(), ldap_next_message(), ldap_first_entry, ldap_next_entry, ldap_first_reference and ldap_next_reference.

Errors

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

If an error occurs in ldap_count_messages(), -1 is returned, and ldap_get_errno() can be used to obtain the error code. See "LDAP_ERROR" on page 50 for a description of possible error codes.

See also

ldap, ldap_result, ldap_first_entry, ldap_next_entry, ldap_first_reference, ldap_next_reference, ldap_get_errno, ldap_msgtype.

LDAP_MODIFY

ldap_modify
ldap_modify_ext
ldap_modify_s
ldap_modify_ext_s
ldap_mods_free

Purpose

Perform various LDAP modify operations.

Synopsis

```
#include <ldap.h>


 typedef struct ldapmod {
        int mod_op;
        char *mod_type;
        union {
        char **modv_strvals;
        struct berval **modv_bvals;
        } mod_vals;
    } LDAPMod;
    #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_ext(
        LDAP           *ld,
        const char     *dn,
        LDAPMod        *mods[],
        LDAPControl    **serverctrls,
        LDAPControl    **clientctrls,
        int            *msgidp)

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

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

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

Input parameters

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

dn Specifies the Distinguished Name (DN) of the entry to be modified. See Appendix B, "LDAP distinguished names" on page 181 for more information about DNs.

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

freemods
Specifies whether or not the mods pointer is to be freed, in addition to the NULL-terminated array of mod structures.

serverctrls
Specifies a list of LDAP server controls. This parameter can be set to NULL. See "LDAP controls" on page 45 for more information about server controls.

clientctrls
Specifies a list of LDAP client controls. This parameter can be set to NULL. See "LDAP controls" on page 45 for more information about client controls.

Output parameters

msgidp
This result parameter is set to the message ID of the request if the ldap_modify_ext() call succeeds.

Usage

The various modify APIs are used to perform an LDAP modify operation. DN is the distinguished name of the entry to modify, and mods is a NULL-terminated array of modifications to make to the entry. Each element of the mods array is a pointer to an LDAPMod structure.

The mod_op field is used to specify the type of modification to perform and must be one of the following:

- LDAP_MOD_ADD (0x00)
- LDAP_MOD_DELETE (0x01)
- LDAP_MOD_REPLACE (0x02)

This field also indicates the type of values included in the mod_vals union. For binary data, you must also logically OR the operation type with LDAP_MOD_BVALUES (0x80). This indicates that the values are specified in a NULL-terminated array of struct berval structures. Otherwise, the mod_values are used, that is, the values are assumed to be a NULL-terminated array of NULL-terminated character strings.

The mod_type field specifies the name of attribute to add, modify or delete.

The mod_vals field specifies a pointer to a NULL-terminated array of values to add, modify or delete. Only one of the mod_values or mod_bvalues variants must be used, with mod_bvalues being selected by ORing the mod_op field with the constant LDAP_MOD_BVALUES.

mod_values is a NULL-terminated array of strings. Since the ldap_add() API converts the string from the local code page to UTF-8, the strings must be in the local code page if the LDAP_OPT_UTF8_IO option has been set to LDAP_UTF8_XLATE_ON for the connection (). If the UTF-8 translation option is not set, the array of strings must be composed of NULL-terminated UTF-8 strings (note that US-ASCII is a proper subset of UTF-8).

mod_bvalues is a NULL-terminated array of berval structures that can be used to pass binary values such as images.

For LDAP_MOD_ADD modifications, the given values are added to the entry, creating the attribute if necessary.

For LDAP_MOD_DELETE modifications, the given values are deleted from the entry, removing the attribute if no values remain. If the entire attribute is to be deleted, the mod_values field must be set to NULL.

For LDAP_MOD_REPLACE modifications, the attribute has the listed values after the modification, having been created if necessary, or removed if the mod_vals field is NULL.

All modifications are performed in the order in which they are listed.

The ldap_modify_ext() API initiates an asynchronous modify operation and returns the constant LDAP_SUCCESS if the request was successfully sent, or another LDAP error code if not. If successful, ldap_modify_ext() places the message ID of the request in *msgidp. A subsequent call to ldap_result() can be used to obtain the result of the operation. Once the operation has completed, ldap_result() returns the status of the operation in the form of an error code. The error code indicates if the operation completed successfully. The ldap_parse_result() API checks the error code in the result.

The ldap_modify() API initiates an asynchronous modify operation and returns the message ID of this operation. A subsequent call to ldap_result(), can be used to obtain the result of the modify. In case of error, ldap_modify() returns -1, setting

the session error parameters in the LDAP structure appropriately, which can be obtained by using ldap_get_errno(). See “LDAP_ERROR” on page 50 for more details.

The synchronous ldap_modify_ext_s() and ldap_modify_s() APIs both return the result of the operation, either the constant LDAP_SUCCESS if the operation was successful, or another LDAP error code if it was not.

The ldap_modify_ext() and ldap_modify_ext_s() APIs support LDAP V3 server controls and client controls.

ldap_modify_s() returns the LDAP error code resulting from the modify operation. This code can be interpreted by ldap_perror() or ldap_err2string().

The ldap_modify() operation works the same way as ldap_modify_s(), except that it is asynchronous, returning the message ID of the request it initiates, or -1 on error. The result of the operation can be obtained by calling ldap_result().

ldap_mods_free() can be used to free each element of a NULL-terminated array of LDAPMod structures. If freemods is non-zero, the mods pointer is freed as well.

Errors

ldap_modify_s() and ldap_modify_ext_s() return the resulting LDAP error code from the modify operation.

ldap_modify() and ldap_modify_ext() return -1 instead of a valid msgid if an error occurs, setting the session error in the LD structure, which can be obtained by using ldap_get_errno(). See “LDAP_ERROR” on page 50 for more details.

See also

ldap, ldap_error, ldap_add

LDAP_PARSE_RESULT

ldap_parse_result

ldap_parse_sasl_bind_result

ldap_parse_extended_result

Purpose

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

Synopsis

```
#include <ldap.h>


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

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

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

Input parameters

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

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

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

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

errmsgp
Specifies a pointer to a result parameter that is filled in with the contents of the error message from the LDAPMessage message. The error message string must be freed by calling ldap_memfree().

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

resultoidp
This result parameter specifies a pointer which is set to point to an allocated, dotted-OID text string returned from the server. This string must be disposed of using the ldap_memfree() API. If no OID is returned, *resultoidp is set to NULL.

resultdatap
This result parameter specifies a pointer to a berval structure pointer that is set to an allocated copy of the data returned by the server. This struct berval must be disposed of using ber_bvfree(). If no data is returned, *resultdatap is set to NULL.

serverctrlsp
Specifies a pointer to a result parameter that is filled in with an allocated

array of controls copied out of the LDAPMessage message. The control array must be freed by calling ldap_controls_free().

freeit Specifies a boolean value that determines if the LDAP result (as specified by res) is to be freed. Any non-zero value results in res being freed after the requested information is extracted. The ldap_msgfree() API can be used to free the result at a later time.

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

err Specifies an LDAP error code, used as input to ldap_err2string(), so that a text description of the error can be obtained.

Usage

The ldap_parse_result() API is used to:

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

The ldap_parse_sasl_bind_result() API is used to obtain server credentials, as a result of an attempt to perform mutual authentication.

Both of the ldap_parse_*_result() APIs ignore messages of type LDAP_RES_SEARCH_ENTRY and LDAP_RES_SEARCH_REFERENCE when looking for a result message to parse. They both return LDAP_SUCCESS if the result was successfully located and parsed, and an LDAP error code if not successfully parsed.

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

Errors

The parse routines return an LDAP error code if they encounter an error parsing the result.

See “LDAP_ERROR” on page 50 for a list of the LDAP error codes.

See also

ldap, ldap_error, ldap_result

LDAP_PLUGIN_REGISTRATION

ldap_register_plugin

ldap_query_plugin

ldap_free_query_plugin

Purpose

Described here are LDAP routines that:

- Register an LDAP client plug-in.
- Obtain information about plug-ins that have been registered by the application, as well as plug-ins that are defined in ldap.conf.
- Free the array of plug-in information returned from the ldap_query_plugin() AP.

Synopsis

```
#include <ldap.h>


int ldap_register_plugin(
       LDAP_File_Plugin_Info *plugin_info)

int ldap_query_plugin(
       LDAP_File_Plugin_Info  plugin_infop )

int ldap_free_query_plugin(
       LDAP_File_Plugin_Info  ***plugin_infop )

typedef struct ldap_file_plugin_info {
    char    *type;              /* plugin type               */
    char    *subtype;           /* plugin subtype            */
    char    *path;              /* path to plugin library    */
    char    *init;              /* initialization routine    */
    char    *paramlist;         /* plugin parameter list     */
} LDAP_File_Plugin_Info;
```

Input parameters

plugin_info

A structure that contains information about a specific type of SASL plug-in. An instance of the structure contains the following fields:

type NULL-terminated string that defines plug-in type. The only type currently supported is sasl.

subtype

NULL-terminated string that specifies the subtype of plug-in being registered. When type=sasl, the subtype is used to specify the SASL mechanism supported by the plug-in. For example, fingerprint might be specified for any SASL plug-in that supports the fingerprint mechanism. For the cram-md5 mechanism, use LDAP_MECHANISM_CRAM_MD5.

path NULL-terminated string that specifies the path to the plug-in's shared library. The plug-in path can be a fully-qualified path including file name, or just the file name with or without the file extension. If just the file name is supplied, the LDAP library attempts to find it using standard operating system search criteria.

init NULL-terminated string that specifies the initialization routine for the plug-in. If NULL, the name of the initialization routine is assumed to be ldap_plugin_init.

parmlist
NULL-terminated string that specifies arbitrary parameter information that is used by the plug-in. For example, if the plug-in needs to access a remote security server, the host name of the remote security server can be supplied as a value in the parameter list.

plugin_infop
Specifies the address that points to a NULL-terminated array of LDAP_Plugin_Info structures. Each LDAP_Plugin_Info structure defined in the list contains information about a registered plug-in. For example:

```
LDAP_File_Plugin_Info  **plugin_infop;

            rc = ldap_query_plugin (&plugin_infop);
```

plugin_infop
Specifies the address of a NULL-terminated array of plug-in information structures to be freed.

Output parameters

plugin_infop
Upon successful return from ldap_query_plugin(), plugin_infop points to a NULL-terminated array of LDAP_Plugin_Info pointers. If there are no plug-ins registered, the plugin_infop data structure is set to NULL and no memory allocated.

Usage

Two mechanisms are available for making an LDAP client plug-in known to the LDAP library:

- As defined in the ldap.conf file.
- The plug-in has been explicitly registered by the application, using the ldap_register_plugin() API.

An application can override the definition of a plug-in in ldap.conf by using the ldap_register_plugin() API. A plug-in is uniquely identified by the combination of its type and subtype. For example, an application can choose to use its own cram-md5 plug-in (as defined in ldap.conf) by invoking ldap_register_plugin() and defining another shared library with type=″sasl″ and subtype=″cram-md5″. Note that plug-ins registered with the ldap_register_plugin() API are defined for the application. In this example, other applications still use the default cram-md5 plug-in.

Finding the Plug-in library

When a plug-in is not explicitly registered by the application with the ldap_register_plugin() API, the LDAP library must find the appropriate plug-in shared library. To find information about the plug-in, the LDAP library must find the ldap.conf file. Note that the attempt to locate ldap.conf is made on behalf of the application in whichever of the following events occurs first:

- The ldap_register_plugin() API is invoked.
- The ldap_sasl_bind_s() API is invoked.

After the ldap.conf file is accessed, all information in the file is stored internally for subsequent use. The file is not re-accessed until the application is restarted. However, additional use of the ldap_register_plugin() API can be used by the application to add additional plug-in definitions, or to override definitions obtained from ldap.conf.

The ldap.conf file: The ldap.conf file contains information required to load and initialize default plug-ins. It can also include additional plug-in-specific configuration information. The following might be defined for each plug-in in the ldap.conf file:

- The plug-in type (for example, sasl)
- The plug-in subtype (for example, mechanism, if type=sasl)
- The path to the plug-in shared library
- The plug-in's initialization routine
- The user-defined parameter string

The ldap.conf file might contain one or more records, each defining this information for a plug-in. Each record takes the following form:

```
plugin type  subtype  path  init-routine parameters
```

For example:

```
#
#   keyword type   subtype          path               init      parameters
#
    plugin  sasl   CRAM-MD5  ldap_plugin_sasl_cram-md5 ldap_plugin_init
    plugin  sasl   fpauth    x:\security\fplib         fpinit    parm2 parm3
    plugin  sasl   hitech    hitechlib                 hitekinit parm5 parm6
```

This example defines three plug-ins (CRAM-MD5, fpauth and hitek) along with associated information.

Note: If the extension is omitted, then an appropriate extension is assumed for the platform, for example, **.a** on the AIX operating system or **.dll** on a Windows operating system. If the fully-qualified path is omitted, then standard OS search rules are applied.

Lines beginning with a number sign (#) are ignored.

The algorithm used to locate ldap.conf is platform specific:

- On a Unix system, the following search order is used:
  1. Query the environment variable IBMLDAP_CONF for the path to ldap.conf.
  2. Look for ldap.conf in the /etc directory.
- On a Windows system, the following search order is used:
  1. Query the environment variable IBMLDAP_CONF for the path to ldap.conf.
  2. Look in current directory for ldap.conf.
  3. Look for lda32p.conf in the /etc directory under the LDAP install directory, for example, c:\Program Files\IBM\LDAP\etc.

If the definition for a SASL plug-in isn't available, the LDAP library encodes the SASL bind and transmits it directly to the LDAP server, bypassing the plug-in facility.

Errors

These routines return an LDAP error code when an error is encountered. To obtain a string description of the LDAP error, use the ldap_err2string() API.

See also

ldap, ldap_error

LDAP_RENAME

ldap_rename

ldap_rename_s

ldap_modrdn (deprecated)

ldap_modrdn_s (deprecated)

Purpose

Perform an LDAP rename operation.

Synopsis

```
#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)

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)
```

Input parameters

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

dn Specifies the DN of the entry whose DN is to be changed. When specified

with the deprecated ldap_modrdn and ldap_modrdn_s APIs, dn specifies the DN of the entry whose RDN is to be changed.

newrdn
Specifies the new RDN given to the entry.

newparent
Specifies the new parent, or superior entry. If this parameter is NULL, only the RDN of the entry is changed. The root DN can be specified by passing a zero length string, "". The newparent parameter is always NULL when using version 2 of the LDAP protocol; otherwise the server's behavior is undefined.

Note: Only NULL is supported by IBM Directory Server version 4.1.

deleteoldrdn
Specifies an integer value. When set to 1, the old RDN value is to be deleted from the entry. When set to 0, the old RDN value must be retained as a non-distinguished value. With respect to the ldap_rename and ldap_rename_s APIs, this parameter only has meaning if newrdn is different from the old RDN.

serverctrls
Specifies a list of LDAP server controls. This parameter can be set to NULL. See "LDAP controls" on page 45 for more information about server controls.

clientctrls
Specifies a list of LDAP client controls. This parameter can be set to NULL. See "LDAP controls" on page 45 for more information about client controls.

Output parameters

msgidp
This result parameter is set to the message ID of the request if the ldap_rename() call succeeds.

Usage

In LDAP V2, the ldap_modrdn() and ldap_modrdn_s() APIs were used to change the name of an LDAP entry. They can be used to change the least significant component of a name (the RDN or relative distinguished name) only. LDAP V3 provides the Modify DN protocol operation that allows more general name change access. The ldap_rename() and ldap_rename_s() routines are used to change the name of an entry, and the use of the ldap_modrdn() and ldap_modrdn_s() routines is deprecated.

The ldap_rename() API initiates an asynchronous modify DN operation and returns the constant LDAP_SUCCESS if the request was successfully sent, or another LDAP error code if not. If successful, ldap_rename() places the message ID of the request in *msgidp. A subsequent call to ldap_result() can be used to obtain the result of the operation. After the operation has completed, ldap_result() returns the status of the operation in the form of an error code. The error code indicates if the operation completed successfully. The ldap_parse_result() API is used to check the error code in the result.

Similarly, the ldap_modrdn() API initiates an asynchronous modify RDN operation and returns the message ID of the operation. A subsequent call to ldap_result(), can be used to obtain the result of the modify. In case of error, ldap_modrdn() returns

-1, setting the session error parameters in the LDAP structure appropriately, which can be obtained by using ldap_get_errno().

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

The ldap_rename() and ldap_rename_s() APIs both support LDAP V3 server controls and client controls.

The ldap_modrdn() and ldap_modrdn_s() routines perform an LDAP modify RDN operation. They both take dn, the DN of the entry whose RDN is to be changed, and newrdn, the new RDN to give to the entry. ldap_modrdn_s() is synchronous, returning the LDAP error code indicating the success or failure of the operation. In addition, they both take the **deleteoldrdn** parameter which is used as an integer value to indicate whether the old RDN values must be deleted from the entry or not.

Errors

The synchronous version of this routine returns an LDAP error code, either LDAP_SUCCESS or an error code if there was an error. The asynchronous version returns -1 in case of an error. If the asynchronous API is successful, ldap_result is used to obtain the results of the operation. See "LDAP_ERROR" on page 50 for more details.

See also

ldap, ldap_error ldap_result

LDAP_RESULT

ldap_result
ldap_msgtype
ldap_msgid

Purpose

Wait for the result of an asynchronous LDAP operation, obtain LDAP message types, or obtain the message ID of an LDAP message.

Synopsis

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


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

int ldap_msgtype(
       LDAPMessage       *msg)

int ldap_msgid(
       LDAPMessage       *msg)
```

Input parameters

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

msgid Specifies the message ID of the operation whose results are to be returned. The parameter can be set to LDAP_RES_ANY if any result is desired.

all This parameter only has meaning for search results. For search results, use **all** to specify how many search result messages are returned in a single call to ldap_result(). Specify LDAP_MSG_ONE to retrieve one search result message at a time. Specify LDAP_MSG_ALL to request that all results of a search be received. ldap_result() waits until all results are received before returning all results in a single chain. Specify LDAP_MSG_RECEIVED to indicate that all results retrieved so far are to be returned in the result chain.

timeout
Specifies how long in seconds to wait for results to be returned from ldap_result, as identified by the supplied msgid. A NULL value causes ldap_result() to wait until results are available. To poll, the timeout parameter is non-NULL, pointing to a zero-valued timeval structure.

msg Specifies a pointer to a result, as returned from ldap_result(), ldap_search_s(), ldap_search_st() or ldap_search_ext().

Output parameters

result Contains the result of the asynchronous operation identified by msgid. This result is passed to the LDAP parsing routines, such as ldap_first_entry().

If ldap_result() is unsuccessful, it returns -1 and sets the appropriate LDAP error ldap_get_errno(). If ldap_result() times out, it returns 0. If successful, it returns one of the following result types:

```
#define LDAP_RES_BIND             0x61L
#define LDAP_RES_SEARCH_ENTRY     0x64L
#define LDAP_RES_SEARCH_RESULT    0x65L
#define LDAP_RES_MODIFY           0x67L
#define LDAP_RES_ADD              0x69L
#define LDAP_RES_DELETE           0x6bL
#define LDAP_RES_MODRDN           0x6dL
#define LDAP_RES_COMPARE          0x6fL
#define LDAP_RES_SEARCH_REFERENCE 0X73L
#define LDAP_RES_EXTENDED         0X78L
#define LDAP_RES_ANY              (-1L)
```

Usage

The ldap_result() routine is used to wait for and return the result of an operation previously initiated by one of the LDAP asynchronous operation routines, for example, ldap_search(), ldap_modify(), and so forth. These routines return a msgid that uniquely identifies the request. The msgid can then be used to request the result of a specific operation from ldap_result().

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

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

Errors

ldap_result() returns 0 if the timeout expires, and -1 if an error occurs. The ldap_get_errno() routine can be used to get an error code.

Notes

This routine allocates memory for results that it receives. The memory can be deallocated by calling ldap_msgfree().

See also

ldap, ldap_search

LDAP_SEARCH

ldap_search

ldap_search_s

ldap_search_ext

ldap_search_ext_s

ldap_search_st

Purpose

Perform various LDAP search operations.

Synopsis

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


int ldap_search(
       LDAP           *ld,
       const char     *base,
       int            scope,
       const char     *filter,
       char           *attrs[],
       int            attrsonly)

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

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

int ldap_search_ext_s(
```

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

int ldap_search_st(
        LDAP           *ld,
        const char     *base,
        int            scope,
        const char     *filter,
        char           *attrs[],
        int            attrsonly,
        struct timeval *timeout,
        LDAPMessage    **res)
```

Input parameters

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

base Specifies the DN of the entry the search starts.

scope Specifies the scope of the search. It can be LDAP_SCOPE_BASE (to search the object itself), or LDAP_SCOPE_ONELEVEL (to search the object's immediate children), or LDAP_SCOPE_SUBTREE (to search the object and all its descendants).

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

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

The '~=' construct is used to specify approximate matching. The representation for *<attributetype>* and *<attributevalue>* are as described in "RFC 2252, LDAP V3 Attribute Syntax Definitions". In addition, *<attributevalue>* can be a single * to achieve an attribute existence test, or can contain text and asterisks (*) interspersed to achieve substring matching.

For example, the filter "mail=*" finds any entries that have a mail attribute. The filter "mail=*@student.of.life.edu" finds any entries that have a mail attribute ending in the specified string. To put parentheses in a filter, escape them with a backslash (\ ) character. See "RFC 2254, A String Representation of LDAP Search Filters" for a more complete description of allowable filters.

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

attrsonly
Specifies attribute information. Attrsonly must be set to 1 to request attribute types only. Set to 0 to request both attribute types and attribute values.

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

timeout
The ldap_search_st() API specifies the local search timeout value. The ldap_search_ext() and ldap_search_ext_s() APIs specify both the local search timeout value and the operation time limit that is sent to the server within the search request.

serverctrls
Specifies a list of LDAP server controls. This parameter can be set to NULL. See "LDAP controls" on page 45 for more information about server controls.

clientctrls
Specifies a list of LDAP client controls. This parameter can be set to NULL. See "LDAP controls" on page 45 for more information about client controls.

Output parameters

res Contains the result of the asynchronous operation identified by msgid, or returned directly from ldap_search_s() or ldap_search_ext_s(). This result is passed to the LDAP parsing routines (see "LDAP_RESULT" on page 84).

msgidp
This result parameter is set to the message ID of the request if the ldap_search_ext() call succeeds.

Usage

These routines are used to perform LDAP search operations.

The ldap_search_ext() API initiates an asynchronous search operation and returns the constant LDAP_SUCCESS if the request was successfully sent, or another LDAP error code if not.

If successful, ldap_search_ext() places the message ID of the request in *msgidp. Use a subsequent call to ldap_result() to obtain the results from the search.

Similar to ldap_search_ext(), the ldap_search() API initiates an asynchronous search operation and returns the message ID of this operation. If an error occurs, ldap_search() returns -1, setting the session error in the LD structure, which can be obtained by using ldap_get_errno(). If successful, use a subsequent call to ldap_result() to obtain the results from the search.

The synchronous ldap_search_ext_s(), ldap_search_s(), and ldap_search_st() functions all return the result of the operation, either the constant LDAP_SUCCESS if the operation was successful, or another LDAP error code if the operation was not successful. See "LDAP_ERROR" on page 50 for more information about possible errors and how to interpret them. If any entries are returned from the

search, they are contained in the **res** parameter. This parameter is opaque to the caller. Entries, attributes, values, and so forth, must be extracted by calling the result parsing routines. The results contained in res must be freed when no longer in use by calling ldap_msgfree().

The ldap_search_ext() and ldap_search_ext_s() APIs support LDAP V3 server controls, client controls, and allow varying size and time limits to be easily specified for each search operation. The ldap_search_st() API is identical to ldap_search_s(), except that it requires an additional parameter specifying a local timeout for the search.

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

LDAP_OPT_SIZELIMIT
A limit on the number of entries returned from the search. 0 means no limit. Note that the value from the session handle is ignored when using the ldap_search_ext() or ldap_search_ext_s() functions.

LDAP_OPT_TIMELIMIT
A limit on the number of seconds to spend on the search. Zero means no limit.

Note: The value from the session handle is ignored when using the ldap_search_ext() or ldap_search_ext_s() functions.

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

These options are set and queried using the ldap_set_option() and ldap_get_option() APIs.

Reading an entry

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

Listing the children of an entry

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

Errors

ldap_search_s(), ldap_search_ext_s and ldap_search_st() return the LDAP error code from the search operation.

ldap_search() and ldap_search_ext() return -1 instead of a valid msgid if an error occurs, setting the session error in the LD structure, which can be obtained by using ldap_get_errno().

See “LDAP_ERROR” on page 50 for more details.

Notes

These routines allocate storage returned by the res parameter. Use ldap_msgfree() to free this storage.

See also

ldap, ldap_result, ldap_error

LDAP_SERVER_INFORMATION IN DNS

ldap_server_locate

ldap_server_free_list

ldap_server_conf_save

Purpose

These LDAP APIs are provided to perform the following operations:

- Use LDAP server information published in the Domain Name System (DNS) to locate one or more LDAP servers, and associated information. Server information is returned as a linked list of server information structures.
- Free all storage associated with a linked list of server information structures.
- Store information about one or more LDAP servers in a local configuration repository. The local configuration can be used to mimic information that can also be published in DNS.

Synopsis

```
#include <ldap.h>


int ldap_server_locate (
       LDAPServerRequest *server_request,
       LDAPServerInfo    **server_info_listpp);

int  ldap_server_free_list(
        LDAPServerInfo *server_info_listp);

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

typedef struct LDAP_Server_Request {
    int     search_source;      /* Source for server info     */
#define LDAP_LSI_CONF_DNS  0    /* Config first, then DNS (def)*/
#define LDAP_LSI_CONF_ONLY 1    /* Local Config file only     */
#define LDAP_LSI_DNS_ONLY  2    /* DNS only                   */
    char    *conf_filename      /* pathname of config file    */
    int     reserved;           /* Reserved, set to zero      */
    char    *service_key;       /* Service string             */
    char    *enetwork_domain;   /* eNetwork domain (eDomain)  */
    char    **name_servers;     /* Array of name server addrs */
    char    **dns_domains;      /* Array of DNS domains       */
    int     connection_type;    /* Connection type            */
```

```
#define LDAP_LSI_UDP_TCP 0      /* Use UDP, then TCP (default)*/
#define LDAP_LSI_UDP 1          /* Use UDP only               */
#define LDAP_LSI_TCP 2          /* Use TCP only               */
    int     connection_timeout; /* connect timeout (seconds)  */
    char    *DN_filter;         /* DN suffix filter           */
    char    *proto_key          /* Symbolic protocol name     */
    unsigned char reserved2[60]; /* reserved fields, set to 0 */
} LDAPServerRequest;


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

Input parameters

server_request

Specifies a pointer to an LDAPServerRequest structure which must be initialized to zero before setting specific parameters. This ensures that defaults are used when a parameter is not explicitly set. If the default behavior is desired for all possible input parameters, simply set server_request to NULL. This is equivalent to setting the LDAPServerRequest structure to zero. Otherwise, supply the address of the LDAPServerRequest structure, containing the following fields:

search_source

Specifies where to find the server information.

1. Access the local LDAP DNS configuration file. If the file is not found, or the file does not contain information for a combination of the service_key, enetwork_domain and any of the DNS domains as specified by the application, then access DNS.
2. Search the local LDAP DNS configuration file only.
3. Search DNS only.

conf_filename

Specifies an alternative configuration filename. Specify NULL to get the default filename and location.

service_key

Specifies the search key, for example, the service name string to be used when obtaining a list of Service records (SRV), pseudo-SRV Text records (TXT) or CNAME alias records from DNS. If not specified, the default is ″ldap.″

Note: Standards are moving towards the use of an underscore (_) as a prefix for service name strings. Over time, it is expected that ″_ldap″ is the preferred service name string for publishing LDAP services in DNS. If the application doesn't specify service_key and no entries are returned using the default ldap service name, the search is automatically rerun using ″_ldap″ as the service name. As an alternative, the application can explicitly specify ″_ldap″ as the service name, and the search is directed specifically at DNS SRV records that use ″_ldap″ as the service name.

enetwork_domain

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

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

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

For example, if:

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

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

If enetwork_domain is set to zero, the following steps are taken to determine the enetwork_domain:

- The locally configured default, if set, is used.
- If a locally configured default is not set, then a platform-specific value is used. On a Windows NT operating system, the user's logon domain is used.
- If a platform-specific eNetwork domain is not defined, then the eNetwork domain component in the DNS value is omitted. In the above example, this results in the following string being used: ldap.midwest.tcp.acme.com.

If enetwork_domain is set to a NULL string, then the eNetwork domain component in the DNS value is omitted. This might be useful for finding a default eNetwork domain when a specific eNetwork domain is not known.

Note: If the search is performed with a non-NULL value for enetwork_domain, and the search fails, the search is issued again with a NULL enetwork_domain, using the specified service_key, which defaults to ldap. The second search with NULL enetwork_domain is attempted after a complete search is concluded without results. For example, if search_source is set to the default LDAP_LSI_CONF_DNS, then the first search is not considered to be complete until both the local configuration and DNS have been queried. If both of these searches fail, then both the local configuration and DNS are re-queried with a NULL enetwork_domain. The intent is to find a set of LDAP servers that are published under the default service key, that is, ldap, when nothing can be found published under ldap.enetwork_domain. The application can determine if the located servers are published in an enetwork_domain by examining the lsi_query_key field, as returned in the server_info_list structures returned on the ldap_server_locate() API. If the returned lsi_query_key consists solely of the specified service_key, then the located servers were not published in DNS with the specified enetwork_domain.

name_servers
Specifies a NULL-terminated array of DNS name server IP address in dotted decimal format, for example, 122.122.33.49. If not specified, the locally configured DNS name servers are used.

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

Note: The domain names supplied here can take the following forms:

- austin.ibm.com (standard DNS format)
- cn=fred, ou=accounting, dc=austin, dc=ibm, dc=com

With respect to providing a domain name, these are equivalent. Both result in a domain name of austin.ibm.com. This approach makes it easier for an application to locate LDAP servers for binding (based on a user name space mapped into the DNS name space). See "DNS domains and configuration file" on page 96 for more information.

connection_type
Specifies the type of connection to use when communicating with the DNS name server. The following options are supported:

- Use UDP first. If no response is received, or data truncation occurs, then use TCP.
- Only use UDP.
- Only use TCP.

If set to zero, the default is to use UDP first (then TCP).

UDP is the preferred connection type, and typically performs well. You might want to consider using TCP/IP if:

- The amount of data being returned does not fit in the 512-byte UDP packet.
- The transmission and receipt of UDP packets turns out to be unreliable. This might depend on network characteristics.

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

DN_filter
Specifies a Distinguished Name to be used as a filter, for selecting candidate LDAP servers based on the server's suffixes. If the most significant portion of the DN is an exact match with a server's suffix (after normalizing for case), an LDAPServerInfo structure is returned for the server/suffix combination. If it doesn't match, an LDAPServerInfo structure is not returned for the server/suffix combination.

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

Note: Standards are moving towards the use of an underscore (_) as a prefix for the protocol. Over time, it is expected that _tcp becomes the preferred protocol string for publishing LDAP and other services in DNS. If the application doesn't specify protocol_key and no entries are returned using the default tcp protocol key, the search is automatically rerun using _tcp as the protocol. As an alternative, the application can explicitly specify _tcp as the protocol, and the search is directed specifically at DNS SRV records that use _tcp as the protocol.

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

server_info_listpp
Specifies the address that is set to point to a linked list of LDAPServerInfo structures. Each LDAPServerInfo structure defined in the list contains server information obtained from either of the following:

- DNS
- Local configuration

filename
Specifies an alternative configuration filename. Specify NULL to get the default filename and location.

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

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

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

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

Output parameters

Returns 0 if successful. If an error is encountered, an appropriate return code as defined in ldap.h is returned. If successful, the address of a linked-list of LDAPServerInfo structures is returned.

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

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

lsi_port
Integer representation of the LDAP server's port.

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

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

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

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

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

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

lsi_weight
The weight value obtained from the SRV RR or the pseudo-SRV TXT RR. Set to zero if unknown or not available.

lsi_vendor_info
NULL-terminated string obtained from the ldapvendor TXT RR, if defined. It might be used to identify the LDAP server vendor/version information.

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

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

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

Usage

DNS domains and configuration file

The local configuration file can contain server information for combinations of the following:

- Service key (typically set to ldap or _ldap)
- eNetwork domain
- DNS domains

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

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

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

  Also, the application uses the default service key, that is, ldap and specifies sales for the eNetwork domain.
- The configuration file contains server information for austin.ibm.com and miami.ibm.com, with the default service key and eNetwork domain of sales).

- Information is also published in DNS for raleigh.ibm.com, with the default service key and eNetwork domain of sales.
- The search_source parameter is set to LDAP_LSI_CONFIG_DNS, which indicates that both the configuration file and DNS are to be used if necessary.
- The locator API builds a single ordered list of server entries, with the following:
  - Server entries for the austin.ibm.com DNS domain, as extracted from the configuration file.
  - Server entries for the raleigh.ibm.com DNS domain, as obtained from DNS over the network.
  - Server entries for the miami.ibm.com DNS domain, as extracted from the configuration file.

The resulting list of servers contains all the austin.ibm.com servers first, followed by the raleigh.ibm.com servers, followed by the miami.ibm.com servers. Within each group of servers, the entries are sorted by priority and weight.

API usage

These routines are used to perform operations related to finding and saving LDAP server information.

ldap_server_locate()

The ldap_server_locate() API is used to locate one or more suitable LDAP servers. In general, an application uses the ldap_server_locate() API as follows:

- Before connecting to an LDAP server in the enterprise, use ldap_server_locate() to obtain a list of one or more LDAP servers that have been published in DNS or in the local configuration file. Typically, an application can simply use the default request settings by passing a NULL for the LDAPServerRequest parameter. By default, the API looks for server information in the local configuration file first, then moves on to DNS if the local configuration file doesn't exist or has expired.

  Note: If no server entries are found, and the application does not specify the service key (which defaults to ldap), then the ldap_server_locate function runs the complete search again, using the alternative "_ldap" for the service key. The results of this second search, if any, are returned to the application.

- Once the application has obtained the list of servers, it must 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:
  - The client needs to specifically connect using SSL or non-SSL. For each server in the list, the application can query the rootDSE to determine if the server supports a secure SSL port. This is the preferred approach. Alternatively, the application can walk the list until it finds a server entry with the appropriate security type. Note that an LDAP server might be listening on both an SSL and non-SSL port. In this case, the server has two entries in the server list:
  - The client specifically needs to connect to a Master or Replica.
  - The client needs to connect to a server that supports a particular suffix.

Note: Specify DN_filter to filter out servers that do not have a suffix. The DN resides under this suffix. To confirm that a server actually supports the suffix, query the server's rootDSE.

– Some other characteristic associated with the desired server exists, perhaps defined in the ldapinfo string.

• After the client has selected a server, it then issues the ldap_init or ldap_ssl_init API. If the selected server is unavailable, the application is free to move down the list of servers until either it finds a suitable server it can connect to, or the list is exhausted.

ldap_server_free_list()

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

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

ldap_server_conf_save()

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

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

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

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

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

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

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

• UNIX— /etc/ldap_server_info.conf

• Windows NT and Windows 2000— %systemroot%\system32\drivers\etc\ldap_server_info.conf

Format of local configuration file: The following is a sample definition for a local configuration file that is created with the ldap_server_conf_save() API. It is recommended that the file be created with the ldap_server_conf_save() API. However, with careful editing, it can also be created and maintained manually.

Some basic rules for managing this file manually:

- Comment fields must begin with a number sign (#). Comment fields are ignored.
- All parameters are positional.
- The first non-comment line must contain the time-to-live value for the file.

```
#################################################################
# Local LDAP DNS configuration file.
#
# The following line holds the file's expiration time, which is
# a UNIX time_t value (time in seconds since January 1, 1970 UTC).
# A value of 0 indicates that the file will not expire.
#907979782
0
# Each of the following lines in this file represents a known
# LDAP server. The lines have the following format:
#
# service domain host priority weight port replica sec "suffix"
       "vendor info" "general info"
#
# where:
#
#  service= service_key[.eNetwork_domain]
#
#  domain=  DNS domain
#
#  host=    fully qualified DNS name of the LDAP Server host
#
#  priority= target host with the lowest priority is tried first
#
#  weight=  load balancing method.  When multiple hosts have the
#           same priority, the host to be contacted first is determined
#           by the weight value.  Set to 0 if load balancing is not needed.
#
#  port=    The port to use to contact the LDAP Server.
#
#  replica= Use "1" to indicate Master.
#           "2" to indicate Replica.
#
#  sec=     Use "1" to indicate Non-SSL
#           "2" to indicate SSL.
#
#  suffix=  A suffix on the server.
#
#  vendor info= a string that identifies the LDAP server vendor
#
#  general info=    Any informational text you wish to include.
#
ldap     austin.ibm.com ldapserver1.austin.ibm.com 1 1 389 1 1
       "ou=users,o=ibm,c=us" "IBM SecureWay" "phoneinfo"
ldap     austin.ibm.com ldapserver2.austin.ibm.com 1 1 389 2 1
       "ou=users,o=ibm,c=us" "IBM SecureWay" "phoneinfo replica"
ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 636 1 2 "" ""
ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 636 1 2
       "cn=GSO,o=IBM,c=US"
ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 636 1 2
       "ou=Austin,o=IBM,c=US" "IBM" "GSO ePersonbase"
ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 389 1 1 "" ""
ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 389 1 1
       "cn=GSO,o=IBM,c=US"
ldap.gso austin.ibm.com gso3.austin.ibm.com 1 1 389 1 1
       "ou=Austin,o=IBM,c=US" "IBM" "GSO ePersonbase"
ldap.sales raleigh.ibm.com saleshost1.raleigh.ibm.com 1 1 389 1 1
       "dc=raleigh,dc=ibm, dc=com" "IBM" "Sales Marketing"
ldap.sales raleigh.ibm.com saleshost2.raleigh.ibm.com 2 1 389 2 1
```

```
        "dc=raleigh,dc=ibm, dc=com" "IBM" "Sales Marketing Replica"
#
###################################################################
```

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

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

Publishing LDAP server information in DNS

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

- If SRV records are supported by the DNS servers in the enterprise, SRV records can be created that identify the LDAP servers, along with appropriate weighting and priority settings. For more information on SRV records and how they are used, see A. Gulbrandsen, P. Vixie, ″A DNS RR for Specifying the Location of Services (DNS SRV)″, Internet RFC 2782, Troll Technologies, Vixie Enterprises,. February, 2000, which obsoletes RFC 2052.
- TXT records must be associated with the A record of each LDAP server. The TXT records include the LDAP URL records which specify host name, port, base DN and port type, for example, ldap for non-SSL, and ldaps for SSL.
- If SRV records are not being used, the list of available servers must be specified with a set of TXT records which emulate the SRV RR format.

The LDAP server locator API:

- Provides access to a list of LDAP servers. By default, the locator API queries a local configuration file for the required information. If the file was updated with a non-zero time-to-live, and the file has become stale, or the file does not contain the required information, the locator API then accesses DNS. By default, the local configuration file has no time-to-live, and is considered to be good indefinitely.

  Note: The configuration file is designed to hold the same level of information per server that can be obtained from DNS.
- Gathers data relevant to each of the LDAP servers from DNS, using three sequenced algorithms:
  1. SRV records
  2. Pseudo-SRV records (using TXT records)
  3. A CNAME alias referencing a single host's A record

The algorithms are attempted in sequence until results are returned for one of the algorithms. For example, if no SRV records are found, but pseudo-SRV records are found, the list of servers is built from the pseudo-SRV records.

- Builds a list of LDAP servers, with the first server in the list classified as the preferred or default server. Depending on how DNS is used to publish LDAP servers, the preferred LDAP server can actually be a reflection of how the administrator has organized the LDAP information in DNS. The application has access to the additional data that was retrieved from DNS. The additional information for each LDAP server information structure can consist of the following:
  - Host name and port
  - eNetwork domain of the server
  - Fully-qualified DNS domain where the hostname is published
  - Suffix
  - Replication type (master or replica)
  - Security type (SSL or non-SSL)
  - Vendor ID
  - Administrator-defined data

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

As input to the API, the application can supply:

- A list of one or more DNS name server IP addresses. The default is to use the locally configured list of name server addresses. Once an active name server is located, it is used for all subsequent processing.
- The service key. The default is ldap. The service key is used to query DNS for information specific to the LDAP protocol. For example, when searching for SRV records in the austin.ibm.com DNS domain, the search is for ldap.tcp.austin.ibm.com with type=SRV. This example assumes the search does not include an eNetwork domain component. The application can also specify _ldap as the service key and _tcp for the protocol, in which case the search is for _ldap._tcp.austin.ibm.com with type=SRV.
- The name of the eNetwork domain. The eNetwork domain is typically the name used to identify the LDAP user's authentication domain, and to further qualify the search for relevant LDAP servers, as published in the user's DNS domain. For example, when searching for SRV records in the austin.ibm.com DNS domain, with an eNetwork domain of marketing the search is for ldap.marketing.tcp.austin.ibm.com with type=SRV.
- A list of one or more fully-qualified DNS domain names. The default is to use the locally configured domains.

  If multiple domains are supplied, either in the default configuration or explicitly supplied by the application, information is gathered from each DNS domain. The server information returned from the locator API is grouped by DNS domain. If two domains are supplied, for example, austin.ibm.com and raleigh.ibm.com, the entries for LDAP servers published in the austin.ibm.com domain appear first in the list, with the austin.ibm.com servers sorted by

priority and weight. Entries for LDAP servers published in the raleigh.ibm.com domain follow the entire set of austin.ibm.com servers (with the raleigh.ibm.com servers sorted by priority and weight).

Note: All entries returned by the locator API are associated with a single <*service_key*>.<*edomain*> combination.

DNS domain names supplied here can take two forms:

- austin.ibm.com (standard DNS format)
- cn=fred, ou=accounting, dc=austin, dc=ibm, dc=com

With respect to providing a fully-qualified DNS domain name, these are equivalent. Both result in a DNS domain name of austin.ibm.com. This approach makes it easier for an application to locate LDAP servers it needs to bind with, based on a user name space mapped into the DNS name space.

- The connection type (UDP or TCP).
- A DN for comparison against the suffix defined for each LDAP server entry. This string, if supplied, is used as a filter. Only server entries that define a suffix that compares with the DN are returned by the locator API. For example, a DN of "cn=fred, ou=accounting, o=ibm, c=us" matches the first of the following, but not the second:
  - o=ibm, c=us
  - o=tivoli, c=us

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

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

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

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

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

The use of SRV records for finding the address of servers, for a specific protocol and domain, is described in RFC 2052, "A DNS RR for Specifying the Location of Services (DNS SRV)." Correct use of the SRV RR permits the administrator to

distribute a service across multiple hosts within a domain, to move the service from host to host without disruption, as well as to designate certain hosts as primary and others as alternates, or backups, by using a priority and weighting scheme.

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

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

where:

service
Symbolic name of the desired service. By default, the service name or service key is ldap. When used to publish servers that are associated with an eNetwork domain, the service value is derived by concatenating the service key, for example, ldap, with the eNetwork domain name, for example, marketing. In this example, the resulting service is ldap.marketing.

proto Protocol, typically tcp or udp, or _tcp or _udp.

name Domain name associated with the RR.

ttl Time-to-live, standard DNS meaning.

class Standard DNS meaning (for example, IN).

Priority
Target host with lowest number priority must be attempted first.

weight
Load balancing mechanism. When multiple target hosts have the same priority, the chance of contacting one of the hosts first must be proportional to its weight. Set to 0 if load balancing is not necessary.

port Port on the target host for the service.

target Target host name must have one or more A records associated with it.

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

- The service TXT record provides a standard LDAP URL, that is, provides host, port and base DN.
- The ldaptype TXT record identifies whether the LDAP server is a master or replica.
- The ldapvendor TXT record identifies the vendor.

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

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

In combination, the name server might contain the following, which effectively publishes the set of LDAP servers that reside in the marketing eNetwork domain:

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

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

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

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

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

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

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

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

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

Note: ldapmsuffix provides an alternate suffix for the 199.23.45.296 host.

The port specified on the SRV record is ignored if the target host has a TXT record containing an LDAP URL. If the URL is specified without a port, the default port is used (389 for non-SSL, 686 for SSL).

Some rules for constructing strings associated with the TXT records:

- If the string contains white space, the entire string following TXT must be enclosed in double quotes.
- If the string contains characters not supported by DNS, for example, the suffix might contain characters not supported by DNS, an escape is supported, based on the technique described in ″Uniform Resource Locators (URL)″, Internet RFC 1738, December 1994. For example:

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

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

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

Using pseudo-SRV TXT records: If the SRV algorithm does not return any servers, the secondary algorithm is invoked. Instead of looking for SRV records, the lookup routine performs a TXT query using the service name string supplied on ldap_server_locate(), which defaults to ldap.tcp.

The intent is to emulate the scheme provided with SRV records, but using a search for TXT records instead. To duplicate the previous example using TXT records instead of SRV records, the following definition is used:

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

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

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

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

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

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

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

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

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

Note: The pseudo-SRV TXT records, in many cases, can exactly replicate the syntax of SRV records, with the exception that SRV is replaced by TXT. This makes for consistent parsing of the records by the resolver routines, plus it makes it very simple to switch between the two mechanisms when inserting this information into the DNS database. However, some versions of DNS require data associated with the TXT records to be enclosed in double quotes, as follows:

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

The ldap_server_locate() API handles either format.

Using a CNAME alias record: If the pseudo-SRV algorithm does not return any servers, the third algorithm is invoked. Instead of looking for TXT records, the lookup routine performs a standard query using the service name string supplied on ldap_server_locate(), which defaults to ldap.

```
ldap.marketing.tcp    CNAME   ldapm

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

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

Alternative scheme for publishing LDAP server information in DNS

A more recent Internet Engineering Task Force (IETF) draft describes a scheme where service keys and the protocol are prefixed with an underscore (_). See the following internet draft for more information on this new scheme: A. Gulbrandsen, P. Vixie, ″A DNS RR for Specifying the Location of Services (DNS SRV)″, Internet RFC 2052, Troll Technologies, Vixie Enterprises. January 1999.

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

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

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

If all LDAP service information is published within your enterprise this way, the application can choose to not specify service key or protocol, and the ldap_server_locate() API first performs its search using ldap and tcp. The search does not find any entries, and the API automatically runs the search again using _ldap and _tcp for service key and protocol, which returns the information published with the alternative scheme.

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

Errors

ldap_server_locate(), ldap_server_free_list and ldap_server_conf_save() return the LDAP error code resulting from the operation.

See “LDAP_ERROR” on page 50 for more details.

See also

ldap, ldap_error

LDAP_SSL

ldap_ssl_client_init

ldap_ssl_init

ldap_ssl_start (deprecated)

ldap_set_cipher

Purpose

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

Synopsis

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

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

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

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

int ldap_set_cipher(
       LDAP       *ld,
       char       *option)
```

Input parameters

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

host Several methods are supported for specifying one or more target LDAP servers, including the following:

Explicit host list

Specifies the name of the host the LDAP server runs on. The host parameter can contain a blank-separated list of hosts to connect to, and each host might optionally be of the form host:port. If present, the :port overrides the port parameter supplied on ldap_init(), ldap_ssl_init() or ldap_open(). The following are typical examples:

```
ld=ldap_ssl_init ("server1", ldap_port, name);
ld=ldap_ssl_init ("server2:636, ldap_port, name);
ld=ldap_ssl_init ( "server1:636 server2:2000 server3",
        ldap_port, name);
```

Local host

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

Default hosts

If the host parameter is set to ldaps://, the LDAP library attempts

to locate one or more default LDAP servers, with secure SSL ports, using the IBM Directory Server ldap_server_locate() function. The port specified on the call is ignored, because ldap_server_locate() returns the port. For example, the following two are equivalent:

```
ld=ldap_ssl_init ("ldaps://", ldap_port, name);
ld=ldap_ssl_init (LDAPS_URL_PREFIX, LDAPS_PORT, name);
```

Note: ldaps or LDAPS_URL_PREFIX must be used to obtain servers with secure ports. If more than one default server is located, the list is processed in sequence, until an active server is found.

The LDAP URL can include a Distinguished Name, used as a filter for selecting candidate LDAP servers based on the server's suffixes. If the most significant portion of the DN is an exact match with a server's suffix after normalizing for case, the server is added to the list of candidate servers. For example, the following returns default LDAP servers that have a suffix that supports the specified DN only:

```
ld=ldap_ssl_init ("ldaps:///cn=fred, dc=austin, dc=ibm,
        dc=com", LDAPS_PORT, name);
```

In this case, a server that has a suffix of ″dc=austin, dc=ibm, dc=com″ matches. If more than one default server is located, the list is processed in sequence, until an active server is found.

If the LDAP URL contains a host name and optional port, the host is used to create the connection. No attempt is made to locate the default servers, and the DN, if present, is ignored. For example, the following two are equivalent:

```
ld=ldap_ssl_init ("ldaps://myserver", LDAPS_PORT, name);
ld=ldap_ssl_init ("myserver", LDAPS_PORT, name);
```

See "Locating default LDAP servers" on page 69 for more information about the algorithm used to locate default LDAP servers.

Host with privileged port

On platforms that support the rresvport function (typically Unix platforms), if a specified host is prefixed with ″privport://″, then the LDAP library uses the rresvport() function to attempt to obtain one of the reserved ports (512 through 1023), instead of an ephemeral port. The search for a reserved port starts at 1023 and stops at 512. If a reserved port cannot be obtained, the function call fails. For example:

```
 ld=ldap_ssl_init ("privport://server1, ldap_port, name);
ld=ldap_ssl_init ("privport://server2:1200, ldap_port,
        name);
ld=ldap_ssl_init ( "privport://server1:800 server2:2000
        privport://server3", ldap_port, name);  port
```

port Specifies the port number to connect to. If you want the default IANA-assigned SSL port of 636, specify LDAPS_PORT.

keyring

Specifies the name of a key database file (with kdb extension). The key database file typically contains one or more certificates of CAs that are trusted by the client. These types of X.509 certificates are also known as trusted roots. A key database can also be used to store the client's private

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

Default keyring and password

Applications can use the default keyring file, as installed with the LDAP support, by specifying NULL pointers for keyring and keyring_pw. The default keyring file, that is, ldapkey.kdb, and the associated password stash file, that is, ldapkey.sth, are installed in the /lib directory under LDAPHOME, where LDAPHOME is the path to the installed LDAP support. LDAPHOME varies by operating system platform:

- AIX - /usr/ldap
- Solaris - /usr/IBMldapc
- Windows - c:\Program Files\IBM\LDAP

  Note: This is the default install location. The actual LDAPHOME is determined during installation.
- HP-UX - /usr/IBMldap

Applications typically use the default keyring file when the LDAP servers used by the applications are configured with X.509 certificates issued by one of the well-known default CA. A trusted root key is the public key and associated Distinguished Name of a CA. The following trusted roots are automatically defined in the default LDAP key database file (ldapkey.kdb):

- Integrion Certification Authority Root
- IBM World Registry™ Certification Authority
- Thawte Personal Premium CA
- Thawte Personal Freeemail CA
- Thawte Personal Basic CA
- Thawte Premium Server CA
- VeriSign Test CA Root Certificate
- RSA Secure Server Certification Authority
- VeriSign Class 1 Public Primary Certification Authority
- VeriSign Class 2 Public Primary Certification Authority
- VeriSign Class 3 Public Primary Certification Authority
- VeriSign Class 4 Public Primary Certification Authority

Note: Each of these certificates are initially set to be trusted. If the default keyring file cannot be located, this set of trusted roots is also built-in to the LDAP/SSL code, and is used by default.

By modifying the contents of ldapkey.kdb, as located in LDAPHOME\lib, all LDAP applications that use SSL and specify NULL pointers to keyring and keyring_pw use the revised key database without change to each application. There are a variety of reasons for changing or customizing a keyring file, including:

- Adding one or more new trusted roots (that is, adding trust for additional CAs).

- Removing trust. For example, your enterprise might obtain all of its server certificates from VeriSign. In this case, it is appropriate to mark the VeriSign certificates as trusted only.

Note: For the default LDAP keyring file to be generally useful to a set of applications, it needs to be readable by each of the applications. It is not suitable to store client certificates with private keys in a keyring file that is readable by users other than the owner of the private keys. Therefore, it is recommended that client certificates with private keys not be stored in the default LDAP keyring file. They must be stored in keyring files that can be accessed by the appropriate user only. Care must be taken to ensure that local file system permissions are set so that the keyring file and associated stash file, if used, are accessible by the appropriate user only.

The password defined for the default ldapkey.kdb file is **ssl_password**. Use this password when initially accessing the default keyring database with the gsk5ikm utility. This default password is also encrypted into the default keyring password stash file, ldapkey.sth, located in the same directory as ldapkey.kdb. Use the gsk5ikm utility to change the password.

If keyring is specified, a fully-qualified path and filename is recommended. If a filename without a fully-qualified path is specified, the LDAP library looks in the current directory for the file. The key database file specified here must have been created using the gsk5ikm utility.

For more information on using gsk5ikm to manage the contents of a key database, see Chapter 4, “Using GSK5IKM” on page 131.

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

keyring_pw

Specifies the password that is used to protect the contents of the key database. This password is important, particularly when it protects one or more private keys stored in the key database. The password is specified when the key database is initially created, and can be changed using the gsk5ikm utility. In lieu of specifying the password each time the application opens the keyring database, the password can be obtained from a password stash file that contains an encrypted version of the password. The password stash file can be created using the gsk5ikm utility. To obtain the password from the password stash file, specify a NULL pointer for keyring_pw. It is assumed that the password stash file has the same name as the keyring database file, but with an extension of .sth instead of .kdb. It is also assumed that the password stash file resides in the same directory as the keyring database file.

Note: The default keyring file (ldapkey.kdb) is initially configured to have **ssl_password** as its password. This password is also initially configured in the default password stash file (ldapkey.sth).

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

If the LDAP server is configured to perform Server Authentication, a client certificate is not required and name can be set to **NULL**. If the LDAP server is configured to perform Client and Server Authentication, a client certificate is required. name can be set to **NULL** if a default certificate/private key pair has been designated as the default. See Chapter 4, "Using GSK5IKM" on page 131. Similarly, name can be set to **NULL** if there is a single certificate/private key pair in the designated key database.

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

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

Usage

The U.S. government's regulations regarding the export of SDKs which provide support for encryption continue to evolve.

The point of control, with respect to available levels of encryption, is now the application.

Any LDAP application which uses the IBM Directory Server C-Client SDK Version 4.1 with the required level of GSKit 5.0.4 or higher has default access to SSL encryption algorithms.

ldap_ssl_client_init() is used to initialize the SSL protocol stack for an application process. Initialization includes establishing access to the specified key database file. The ldap_ssl_client_init() API must be invoked once per application process, prior to making any other SSL-related LDAP calls, such as ldap_ssl_init(). Once ldap_ssl_client_init() has been successfully invoked, any subsequent invocations return a return code of LDAP_SSL_ALREADY_INITIALIZED. This also means that a particular key database file is effectively bound to an application process. To change the key database, the application or one of its processes must be restarted.

ldap_ssl_environment_init() can be used instead of ldap_ssl_client_init() with the advantage of being able to be called more than once in the same process. Each call creates a new SSL environment which is utilized for subsequent SSL sessions

initiated by calling ldap_ssl_init(). These SSL environments persist as long as the LDAP sessions that were created using them persist.

ldap_ssl_init() is the SSL equivalent of ldap_init(). It is used to initialize a secure SSL session with a server.

Note: The server is not actually contacted until an operation is performed that requires it, allowing various options to be set after initialization.

After the secure connection is established for the LDAP session, all subsequent LDAP messages that flow over the secure connection are encrypted, including the ldap_simple_bind() parameters, until ldap_unbind() is invoked.

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

The LDAP session handle returned by ldap_ssl_init and ldap_init is a pointer to an opaque data type representing an LDAP session. The ldap_get_option() and ldap_set_option() APIs are used to access and set a variety of session-wide parameters. See "LDAP_INIT" on page 59 for more information about ldap_get_option() and ldap_set_option().

Note: When connecting to an LDAP V2 server, one of the ldap_simple_bind() or ldap_bind() calls must be completed before other operations can be performed on the session, with the exception of ldap_set/get_option(). The LDAP V3 protocol does not require a bind operation before performing other operations.

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

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

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

...additional LDAP API calls

rc = ldap_unbind( ld );
```

Note: The sequence of calls for the deprecated APIs is ldap_open/init(), ldap_ssl_start(), followed by ldap_bind().

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

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

See ldap_get/set_option() for more information on setting the ciphers to be used.

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

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

where timeout specifies timeout in seconds. When timeout occurs, SSL again establishes the session keys for the session, for increased security. To specify a specific cipher, or set of ciphers, to be used when negotiating with the server, use ldap_set_option() to define a sequence of ciphers. For example, the following defines a sequence of three ciphers to be used when negotiating with the server. The first cipher that is found to be in common with the server's list of ciphers is used.

ldap_set_cipher is the same as calling ldap_set_option (ld, LDAP_OPT_SSL_CIPHER, option). Either function checks the validity of the input string. The cipher is used when the SSL connection is established by ldap_ssl_init(). See "LDAP_INIT" on page 59 for more information about ldap_set_option.

Options

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

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

The following ciphers are defined in ldap.h:

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

For more information on ldap_set_option, see "LDAP_INIT" on page 59.

Notes

ldapssl.h contains return codes that are specific for ldap_ssl_client_init(), ldap_ssl_init() and ldap_ssl_start().

The SSL versions of these utilities include RSA Security Inc. software.

The ldap_ssl_client_init(), ldap_ssl_init() and ldap_ssl_start() APIs are only supported for the versions of the LDAP library that include the SSL component.

See also

ldap, ldap_open

LDAP_URL

ldap_is_ldap_url
ldap_url_parse
ldap_free_urldesc
ldap_url_search
ldap_url_search_s
ldap_url_search_st

Purpose

LDAP Uniform Resource Locator routines.

Synopsis

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

#include <ldap.h>


int ldap_is_ldap_url(
       char           *url)

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

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

ldap_free_urldesc(
       LDAPURLDesc    *ludp)

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

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

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

Input parameters

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

url Specifies a pointer to the URL string.

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

timeout
Specifies a timeout value for a synchronous search issued by the ldap_url_search_st() routine.

ludp Points to the LDAP URL description, as returned by ldap_url_parse().

Output parameters

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

res Contains the result of the asynchronous operation identified by msgid, as returned from ldap_url_search_s() or ldap_url_search_st(). This result must be passed to the LDAP parsing routines.

Usage

These routines support the use of LDAP URLs. LDAP URLs look like the following:

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

where:
- *hostport* is a host name with an optional :portnumber.
- *dn* is the base DN to be used for an LDAP search operation.
- *attributes* is a comma-separated list of attributes to be retrieved.
- *scope* is one of the following three strings: base, one, or sub. The default is base.
- *filter* is the LDAP search filter as used in a call to ldap_search.

For example:

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

URLs that are wrapped in angle-brackets or preceded by **URL:** or both are also tolerated, including the following forms:

- URL:ldapurl

  For example:

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

- <URL:ldapurl>

  For example:

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

ldap_is_ldap_url() returns a non-zero value if url begins with **ldap://**. It can be used as a quick check for an LDAP URL; the ldap_url_parse() routine is used to extract the various components of the URL.

ldap_url_parse() breaks down an LDAP URL passed in url into its component pieces. If successful, zero is returned, an LDAP URL description is allocated and filled in, and ludpp is set to point to it. If an error occurs, one of these values is returned:

```
LDAP_URL_ERR_NOTLDAP    - URL doesn't begin with "ldap://"
LDAP_URL_ERR_NODN   - URL has no DN (required)
LDAP_URL_ERR_BADSCOPE   - URL scope string is invalid
LDAP_URL_ERR_MEM    - can't allocate memory space
```

ldap_free_urldesc() is called to free an LDAP URL description that was obtained from a call to ldap_url_parse().

ldap_url_search() initiates an asynchronous LDAP search based on the contents of the URL string. This routine acts just like ldap_search except that the search parameters are pulled out of the URL.

ldap_url_search_s() performs a synchronous LDAP search based on the contents of the URL string. This routine acts just like ldap_search_s() except that the search parameters are pulled out of the URL.

ldap_url_search_st() performs a synchronous LDAP URL search with a specified timeout. This routine acts just like ldap_search_st() except that the search parameters are pulled out of the URL.

Notes

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

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

See also

ldap, ldap_search

LDAP_CODEPAGE

ldap_xlate_local_to_utf8
ldap_xlate_utf8_to_local
ldap_xlate_local_to_unicode
ldap_xlate_unicode_to_local
ldap_set_locale
ldap_get_locale
ldap_set_iconv_local_codepage
ldap_get_iconv_locale_codepage
ldap_set_iconv_local_charset
ldap_char_size

Purpose

Functions for managing the conversion of strings between UTF-8 and a local code page.

Synopsis

```
#include <ldap.h>


int  ldap_xlate_local_to_utf8(
       char           *inbufp,
       unsigned long  *inlenp,
       char           *outbufp,
```

```
       unsigned long  *outlenp)

int  ldap_xlate_utf8_to_local(
       char           *inbufp,
       unsigned long  *inlenp,
       char           *outbufp,
       unsigned long  *outlenp)

int  ldap_xlate_local_to_unicode(
       char           *inbufp,
       unsigned long  *inlenp,
       char           *outbufp,
       unsigned long  *outlenp)

int  ldap_xlate_unicode_to_local(
       char           *inbufp,
       unsigned long  *inlenp,
       char           *outbufp,
       unsigned long  *outlenp)

int  ldap_set_locale(
       char           *locale)

char *ldap_get_locale( )

int  ldap_set_iconv_local_codepage
       char           *codepage)

char *ldap_get_iconv_local_codepage( )

int  ldap_set_iconv_local_charset(
       char           *charset)

int  ldap_char_size(
       char           *p)
```

Input parameters

inbufp
A pointer to the address of the input buffer containing the data to be translated

inlenp Length in bytes of the inbufp input buffer

outbufp
A pointer to the address of the output buffer for translated data

outlenp
Length in bytes of the outbufp input buffer

Note: The output buffer must be three times as large as the input buffer if the intent is to translate the entire input buffer in a single call.

charset
Specifies the character set to be used when converting strings between UTF-8 and the local code page. See "IANA character sets supported by platform" on page 185 for the specific charset values that are supported for each operating system platform.

Note: The supported values for charset are the same values supported for the charset tag that is optionally defined in Version 1 LDIF files.

codepage
Specifies a code page or code set for overriding the active code page for

the currently defined locale. See the system documentation for the code pages supported for a particular operating system.

locale Specifies the locale to be used by LDAP when converting to and from UTF-8 or Unicode. If the locale is not explicitly set, the LDAP library uses the application's default locale. To force the LDAP library to use another locale, specify the appropriate locale string.

For applications running on the Windows platform, supported locales are defined in ldaplocale.h. For example, the following is an excerpt from ldaplocale.h and shows the available French locales:

```
/*      French - France                                 */
     #define LDAP_LOCALE_FRFR850            "Fr_FR"
     #define LDAP_LOCALE_FRFRISO8859_1      "fr_FR"
```

For applications running on the AIX operating system, see the locale definitions defined in the "Understanding Locale" chapter of *AIX System Management Guide: Operating System and Devices*. System-defined locales are located in /usr/lib/nls/loc on the AIX operating system. For example, Fr_FR and fr_FR are two system-supported French locales.

For Solaris applications, see the system documentation for the set of system-supported locale definitions.

Note: The specified locale is applicable to all conversions by the LDAP library within the applications address space. The LDAP locale is set or changed only when there is no other LDAP activity occurring within the application on other threads.

p Returns the number of bytes constituting the character pointed to by **p**. For ASCII characters, this is 1. For other character sets, it can be greater than 1.

Output parameters

inbufp
A pointer to the address of the input buffer containing the data to be translated

inlenp Length in bytes of the inbufp input buffer

outbufp
A pointer to the address of the output buffer for translated data

outlenp
Length in bytes of the outbufp input buffer

Note: The output buffer must be three times as large as the input buffer if the intent is to translate the entire input buffer in a single call.

locale When returned from the ldap_get_locale() API, locale specifies the currently active locale for LDAP. See the system documentation for the locales supported for a particular operating system. For applications running in the Windows environment, see ldaplocale.h.

codepage
When returned from ldap_get_iconv_local_codepage() API, codepage specifies the currently active code page, as associated with the currently active locale. See the system documentation for the code pages supported for a particular operating system.

Usage

These routines described in the sections below are used to manage application-level conversion of data between the local code page and UTF-8, which is used by LDAP when communicating with an LDAP V3 compliant server. For more information on the UTF-8 standard, see "UTF-8, a Transformation Format of ISO 10646".

When connected to an LDAP V3 server, the LDAP APIs are designed to accept and return string data UTF-8 encoded. This is the default mode of operation. Alternatively, your application can rely on the LDAP library to convert LDAP V3 string data to and from UTF-8 by using the ldap_set_option() API to set the LDAP_OPT_UTF8_IO option to LDAP_UTF8_XLATE_ON. Once set, the following connection-based APIs, that is, those that accept an ld as input, expect string data to be supplied as input in the local code page, and return string data to the application in the local code page. In other words, the following LDAP routines and related APIs automatically convert string data to and from the UTF-8 wire protocol:

- ldap_add (and family)
- ldap_bind (and family)
- ldap_compare (and family)
- ldap_delete (and family)
- ldap_parse_reference
- ldap_get_dn
- ldap_get_values
- ldap_modify (and family)
- ldap_parse_result
- ldap_rename (and family)
- ldap_search (and family)
- ldap_url_search (and family)

The following APIs are not associated with a connection, and always expect string data, for example, DNs, to be supplied and returned UTF-8 encoded:

- ldap_explode_dn
- ldap_explode_dns
- ldap_explode_rdn
- ldap_server_locate
- ldap_server_conf_save
- ldap_is_ldap_url
- ldap_url_parse
- ldap_default_dn_set

The APIs described in this section provide assistance in converting your application data to and from the locale code page. There are several reasons for using these APIs:

- The application is using one or more of the non-connection oriented APIs, and needs to convert strings to UTF-8 from the local code page before using the APIs.
- The application is designed to send and receive strings as UTF-8 when using the LDAP APIs, but needs to convert selected strings to the local code page before

presenting to the user. When the directory contains heterogeneous data, that is, data is obtained from multiple countries, or locales, this might be the desired approach.

If your application might be extracting string data from the directory that has originated from other countries or locales, design the application with the following considerations in mind:

- Consider splitting your application into a presentation component, and an LDAP worker component.
  - The presentation component is responsible for obtaining data from external sources, for example, graphical user interfaces (GUIs), command-lines, files, and so forth, as well as displaying the data to a GUI, standard out, files, and so forth. This component typically deals with string data that is represented in the local code page.
  - The LDAP worker component is responsible for interfacing directly with the LDAP programming interfaces. The LDAP worker component can be implemented to deal strictly in UTF-8 when handling string data. The default mode of operation for the LDAP library is to handle strings encoded as UTF-8.
  - String conversion between UTF-8 and the local code page occurs when data is passed to and from the presentation component and the LDAP worker component.

  Consider the following scenario:

  The LDAP worker component issues an LDAP search, and returns a list of entries from the directory. To ensure that no data is lost, the default mode is used and the LDAP library does not convert string data. In this case, this means the DNs of the entries returned from the search are represented in UTF-8.

  The application needs to display this list of DNs on a panel, so the user can select the desired entry, and the application then retrieves additional attributes for the selected DN. Since the DN is represented in UTF-8, it must be converted to the local code page prior to display.

  The converted DN might not be a faithful representation of the UTF-8 DN. For example, if the DN was created in China, it can contain Chinese characters. If the application is running in a French locale, certain Chinese characters might not be converted correctly, and are replaced with a replacement character.

  The application can display the converted DN, but certain characters might be displayed as bobs. Assuming there is enough information for the end-user to select the desired DN, the application accesses the LDAP directory with the selected DN to get additional information, for example, a jpeg image so it can display the user's photograph. Since jpeg images might be large, the application is designed to obtain the jpeg attribute after the user selects the specific DN only.

  In order to ensure that the search to get the jpeg attribute using the selected DN works, the search must be done with the original UTF-8 version of the selected DN, not the version of the DN that was converted to the local code page. This implies that the application maintains a correlation between the original UTF-8 version of the DN, and the version that was converted to the local code page.
- If the application is designed to accept user input, generate one or more LDAP searches, then display the information without passing the results back into the

LDAP library. The application can be designed to let the LDAP library perform the conversions, even though some data loss might theoretically occur. Automatic conversion of string data for a specific ld can be enabled by using ldap_set_option() with the LDAP_OPT_UTF8_IO option set to LDAP_UTF8_XLATE_ON.

ldap_char_size returns the number of bytes constituting the character pointed to by p. For ASCII characters, this is 1. For other character sets, it can be greater than 1.

Translate local code page to UTF-8

The ldap_xlate_local_to_utf8() API is used to convert a string from the local code page to a UTF-8 encoding. Since the output string from the conversion process can be larger than the input string, it is strongly recommended that the output buffer be at least twice as large as the input buffer. LDAP_SUCCESS is returned if the conversion is successful.

Translate UTF-8 to local code page

The ldap_xlate_utf8_to_local() API is used to convert a UTF-8 encoded string to the local code page encoding. Since the output string from the conversion process can be larger than the input string, it is strongly recommended that the output buffer be at least twice as large as the input buffer. LDAP_SUCCESS is returned if the conversion is successful.

Note: Translation of strings from a UTF-8 encoding to local code page can result in loss of data when one or more characters in the UTF-8 encoding cannot be represented in the local code page. When this occurs, a substitution character replaces any UTF-8 characters that cannot be converted to the local code page.

Translate local code page to unicode

The ldap_xlate_local_to_unicode() API is used to convert a string from the local code page to the UCS-2 encoding as defined by ISO/IEC 10646-1. This same set of characters is also defined in the UNICODE standard. Since the output string from the conversion process can be larger than the input string, it is strongly recommended that the output buffer be at least twice as large as the input buffer. LDAP_SUCCESS is returned if the conversion is successful.

Translate unicode to local code page

The ldap_xlate_unicode_to_local() API is used to convert a UCS-2-encoded string to the local code page encoding. Since the output string from the conversion process can be larger than the input string, it is strongly recommended that the output buffer be at least twice as large as the input buffer. LDAP_SUCCESS is returned if the conversion is successful.

Note: Translation of strings from a UCS-2 (UNICODE) encoding to local code page can result in loss of data when one or more characters in the UCS-2 encoding cannot be represented in the local code page. When this occurs, a substitution character replaces any UCS-2 characters that cannot be converted to the local code page.

Set locale

The ldap_set_locale() API is used to change the locale used by LDAP for conversions between the local code page and UTF-8 (or Unicode). Unless explicitly set with the ldap_set_locale() API, LDAP uses the application's default locale. To force the LDAP library to use another locale, specify the appropriate locale string. For Unix systems, see the system documentation for the locale definitions. For Windows operating systems, see ldaplocale.h.

Get locale

The ldap_get_locale() API is used to obtain the active LDAP locale. Values that can be returned are system-specific.

Set codepage

The ldap_set_iconv_local_codepage() API is used to override the code page associated with the active locale. See the system documentation for the code pages supported for a particular operating system.

Get codepage

The ldap_get_iconv_local_codepage() API is used to obtain the code page associated with the active locale. See the system documentation for the code pages supported for a particular operating system. See "IANA character sets supported by platform" on page 185 for the specific charset values that are supported for each operating system platform. Note that the supported values for charset are the same values supported for the charset tag that is optionally defined in Version 1 LDIF files.

Japanese and Korean currency considerations

The generally accepted convention for converting the backslash character (\ ) (single byte X'5C') from the Japanese or Korean locale into Unicode is to convert X'5C' to the Unicode yen for Japanese, or the Unicode won for Korean.

To change the default behavior, set the LDAP_BACKSLASH environment variable to YES prior to using any of the LDAP APIs. When LDAP_BACKSLASH is set to YES, the X'5C' character is converted to the Unicode (\ ) , instead of the Japanese yen or Korean won.

Errors

Each of the LDAP user configuration APIs returns a non-zero LDAP return code if an error occurs. See "LDAP_ERROR" on page 50 for more details.

See also

ldap, ldap_error

LDAP_SSL_ENVIRONMENT_INIT

Purpose

ldap_ssl_environment_init() has the same parameters as ldap_ssl_client_int() but can be called more than once. It returns LDAP_SUCCESS or the appropriate LDAP error code. It does not return LDAP_SSL_ALREADY_INITIALIZED. An application that requires SSL connections to different servers can initialize environments in separate calls to this function, with different key database files. The environment created is used by all SSL connections established by calling ldap_ssl_init() until the next call is made to ldap_ssl_environment_init(). Subsequent calls to ldap_ssl_environment_init() do not affect existing SSL connections.

Synopsis

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

int ldap_ssl_environment_init(
```

```
char       *keydatabase,
char       *keydatabase_pw,
int        ssl_timeout,
int        *pSSLReasonCode)
```

where

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

keydatabase_pw
Specifies the password that is used to protect the contents of the key database. This password is important, particularly when it protects one or more private keys stored in the key database. The password is specified when the key database is initially created, and can be changed using the gsk5ikm utility. Instead of specifying the password each time the application opens the key database, the password can be obtained from a password stash file that contains an encrypted version of the password. The password stash file can be created using the gsk5ikm utility. To obtain the password from the password stash file, specify a NULL pointer for keydatabase_pw. It is assumed that the password stash file has the same name as the key database file, but with a .sth extension instead of .kdb. It is assumed that the password stash file resides in the same directory as the key database file.

Note: The default key database file, ldapkey.kdb, is initially configured to have **ssl_password** as its password. This password is also initially configured in the default password stash file (ldapkey.sth).

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

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

LDAP_SORT

ldap_create_sort_keylist
ldap_free_sort_keylist
ldap_create_sort_control
ldap_parse_sort_control

Purpose

Used to request sort of entries returned by the servers that match the filter specified on a search operation.

Synopsis

```
#include <ldap.h>

typedef struct _LDAPsortkey {
     char *attr_type; /* name of attribute */
     char *match_rule_oid; /* OID of matching rule */
     int   reverse_order; /* specifies if attribute
           is sorted in reverse order */
} LDAPsortkey;

int ldap_create_sort_keylist (LDAPsortkey ***sortKeyList,
     const char *sortString);

int ldap_create_sort_control (LDAP *ld, LDAPsortkey
      **sortKeyList, const char isCritical, LDAPControl
      **control);

void ldap_free_sort_keylist (LDAPsortkey **sortKeyList);

int ldap_parse_sort_control (LDAP *ld, LDAPControl
      **serverControls, unsigned long *sortRC, char
      **attribute);
```

Input parameters

ld Specifies the LDAP pointer returned by previous call to ldap_init(), ldap_ssl_init() or ldap_open(). Must not be NULL.

sortString
String with one or more attributes to be used to sort entries returned by the server.

sortKeyList
Pointer to an array of LDAPsortkey structures, which represent attributes that the server uses to sort returned entries. Input when used for ldap_create_sort_control() and ldap_free_sort_keylist().

isCritical
Specifies the criticality of sort on the search. If the criticality of sort is FALSE, and the server finds a problem with the sort criteria, the search continues but entries returned are not sorted. If the criticality of sort is TRUE, and the server finds a problem with the sort criteria, the search does not continue, no sorting is done, and no entries are returned. If the server does not find any problem with the sort criteria, the search and sort continues and entries are returned sorted.

serverControls
A list of LDAP server controls. See “LDAP controls” on page 45 for more information about server controls. These controls are returned to the client when calling the ldap_parse_result() function on the set of results returned by the server.

Output parameters

sortKeyList
Pointer to an array of LDAPsortkey structures, which represent attributes the server uses to sort returned entries. Output when used for ldap_create_sort_keylist().

control
A result parameter that is filled in with an allocated array of one control for the sort function. The control must be freed by calling ldap_control_free().

sortRC
LDAP return code retrieved from the sort results control returned by the server.

attribute
Returned by the server, this is the name of the attribute in error.

Usage

These routines are used to perform sorting of entries returned from the server following an LDAP search operation.

The ldap_create_sort_keylist() function builds a list of LDAPsortkey structures based on the list of attributes included in the incoming string. A sort key is made up of three possible values:

- Name of attribute used to sort entries returned by the server
- OID of a matching rule for that attribute
- Whether or not the sort must be done in reverse order

The syntax of the attributes in the sortString, [-]<attribute name>[:<matching rule OID>], specifies whether or not there is a matching rule OID that must be used for the attribute, and whether or not the attribute must be sorted in reverse order. In the following example sortString, the search results are sorted first by surname and then by given name, with the given name being sorted in reverse (descending order) as specified by the prefixed minus sign (-):

```
sn -givenname
```

Thus, the syntax of the sort parameter is as follows:

```
[-]<attribute name>[:<matching rule OID>]
```

where

- `attribute name` is the name of the attribute you want to sort by.
- `matching rule OID` is the optional OID of a matching rule that you want to use for sorting.
- the minus sign (-) indicates that the results must be sorted in reverse order.

The sortKeyList, output from the ldap_create_sort_keylist() function, can be used as input into the ldap_create_sort_control() function. The sortKeyList is an ordered array of LDAPsortkey structures such that the key with the highest precedence is at the front of the array. The control output form ldap_create_sort_control() function includes the criticality set based on the value of the isCritical flag. This control is added to the list of client controls sent to the server on the LDAP search request.

The ldap_free_sort_keylist() function cleans up all the memory used by the sort key list. This function must be called after the ldap_create_sort_control() function has completed.

When a sort results control is returned by the server, the ldap_parse_sort_control() function can be used to retrieve the values from the control. The function takes as input the server controls returned by the server, and returns the value of the sort control return code and possibly an attribute name if the return code is not LDAP_SUCCESS. If there was an error parsing the sort criteria for the search or there were no entries returned for the search, no sort control is returned to the client.

Errors

The sort routines return an LDAP error code if they encounter an error parsing the result. See "LDAP_ERROR" on page 50 for a list of the LDAP error codes.

Notes

SortString, sortKeyList, controls, serverControls, and attribute must be freed by the caller.

See also

ldap, ldap_search, ldap_parse_result

LDAP_PAGED_RESULTS

ldap_create_page_control

ldap_parse_page_control

Purpose

Used to request simple paged results of entries returned by the servers that match the filter specified on a search operation.

Synopsis

```
#include <ldap.h>

int ldap_create_page_control LDAP_P ((LDAP *ld, unsigned long pageSize,
      struct berval *cookie, const char isCritical, LDAPControl **control));

int ldap_parse_page_control LDAP_P ((LDAP *ld, LDAPControl
      **serverControls, unsigned long *totalCount, struct berval **cookie));
```

Input parameters

ld Specifies the LDAP pointer returned by previous call to ldap_init(), ldap_ssl_init() or ldap_open(). Must not be NULL.

pageSize
Number of entries that are returned for this paged results search request.

cookie Opaque structure returned by the server. No assumptions must be made about the internal organization or value. The cookie is used on subsequent paged results search requests when more entries are to be retrieved from the results set. The cookie must be the value of the cookie returned on the last response returned from the server on all subsequent paged results search requests. The cookie is empty when there are no more entries to be

returned by the server, or when the client abandons the paged results request by sending in a zero page size. Once the paged results search request is completed, the cookie must not be used because it is no longer valid.

isCritical
Specifies the criticality of paged results on the search. Whether the criticality of paged results is TRUE or FALSE, and the server finds a problem with the sort criteria, the search does not continue. If the server does not find any problem with the paged results criteria, the search continues and entries are returned one page at a time.

serverControls
A list of LDAP server controls. See "LDAP controls" on page 45 for more information about server controls. These controls are returned to the client when calling the ldap_parse_result() function on the set of results returned by the server.

Output parameters

control
A result parameter that is filled in with an allocated array of one control for the sort function. The control must be freed by calling ldap_control_free().

totalCount
Estimate of the total number of entries for this search, can be zero if the estimate cannot be provided.

cookie Opaque structure returned by the server. No assumptions must be made about the internal organization or value. The cookie is used on subsequent paged results search requests when more entries are to be retrieved from the results set. The cookie must be the value of the cookie returned on the last response returned from the server on all subsequent paged results search requests. The cookie is empty when there are no more entries to be returned by the server, or when the client abandons the paged results request by sending in a zero page size. Once the paged results search request is completed, the cookie must not be used because it is no longer valid.

Usage

The ldap_create_page_control() function uses the page size and the cookie to build the paged results control. The control output from ldap_create_page_control() function includes the criticality set based on the value of the isCritical flag. This control is added to the list of client controls sent to the server on the ldap search request.

When a paged results control is returned by the server, the ldap_parse_page_control() function can be used to retrieve the values from the control. The function takes as input the server controls returned by the server, and returns a cookie to be used on the next paged results request for this search operation.

Note: If the page size is greater than or equal to the search sizeLimit value , the server ignores the paged results control because the request can be satisfied in a single page. No paged results control value is returned by the server in this case. In all other cases, error or not, the server returns a paged results control to the client.

Errors

The sort routines return an LDAP error code if they encounter an error parsing the result. See "LDAP_ERROR" on page 50 for a list of the LDAP error codes.

Notes

Controls, serverControls, and cookie must be freed by the caller.

See also

ldap, ldap_search, ldap_parse_result

Possible extended error codes returned by LDAP SSL function codes

The following are values returned by all function calls:

- 0 –The task completed successfully. Issued by every function call that completes successfully.
- 1 – The environment or SSL handle is not valid. The specified handle was not the result of a successful open function call.
- 2 – The dynamic link library unloaded (Windows only).
- 3 – An internal error occurred. Report this error to service.
- 4 – Main memory is insufficient to perform the operation.
- 5 – The handle is in an invalid state for operation, such as performing an init operation on a handle twice.
- 6 – Specified key label not found in keyfile.
- 7 – Certificate not received from partner.
- 8 – Certificate validation error.
- 9 – Error processing cryptography.
- 10 – Error validating Abstract Syntax Notation (ASN) fields in certificate.
- 11 – Error connecting to LDAP server.
- 12 – Internal unknown error. Report problem to service.
- 101 – Internal unknown error. Report problem to service.
- 102 – I/O error reading keyfile.
- 103 – Keyfile has an invalid internal format. Re-create keyfile.
- 104 – Keyfile has two entries with the same key. Use iKeyman to remove the duplicate key.
- 105 – Keyfile has two entries with the same label. Use iKeyman to remove the duplicate label.
- 106 – The keyfile password is used as an integrity check. Either the keyfile has become corrupted or the password ID is incorrect.
- 107 – The default key in the keyfile has an expired certificate. Use iKeyman to remove certificates that are expired.
- 108 – There was an error loading one of the GSKdynamic link libraries. Be sure GSK was installed correctly.
- 109 – Indicates that a connection is trying to be made in a gsk environment after the GSK_ENVIRONMENT_CLOSE_OPTIONS has been set to GSK_DELAYED_ENVIRONMENT_CLOSE and gsk_environment_close() function has been called.
- 201 – Neither the password nor the stash-file name was specified, so the key file could not be initialized.

- 202 – Unable to open the key file. Either the path was specified incorrectly or the file permissions did not allow the file to be opened.
- 203 – Unable to generate a temporary key pair. Report this error to service.
- 204 – A User Name object was specified that is not found
- 205 – A Password used for an LDAP query is not correct
- 206 – An index into the Fail Over list of LDAP servers was not correct.
- 301 – Indicates that the GSK environment close request was not properly handled. Cause is most likely due to a gsk_secure_socket*() command being attempted after a gsk_close_environment() call.
- 401 – The system date was set to an invalid value.
- 402 – Neither SSLv2 nor SSLv3 is enabled.
- 403 – The required certificate was not received from partner.
- 404 – The received certificate was formatted incorrectly.
- 405 – The received certificate type was not supported.
- 406 – An IO error occurred on a data read or write.
- 407 – The specified label in the key file could not be found.
- 408 – The specified key file password is incorrect. The key file could not be used. The key file may also be corrupt.
- 409 – In a restricted cryptography environment, the key size is too long to be supported.
- 410 – An incorrectly formatted SSL message was received from the partner.
- 411 – The message authentication code (MAC) was not successfully verified.
- 412 – Unsupported SSL protocol or unsupported certificate type.
- 413 – The received certificate contained an incorrect signature.
- 414 – Incorrectly formatted certificate received from partner.
- 415 – Invalid SSL protocol received from partner.
- 416 – Internal error. Report problem to service.
- 417 – The self-signed certificate is not valid.
- 418 – The read failed. Report this error to service.
- 419 – The write failed. Report this error to service.
- 420 – The partner closed the socket before the protocol completed.
- 421 – The specified V2 cipher is not valid.
- 422 – The specified V3 cipher is not valid.
- 423 – Internal error. Report problem to service.
- 424 – Internal error. Report problem to service.
- 425 – The handle could not be created. Report this internal error to service.
- 426 – Initialization failed. Report this internal error to service.
- 427 – When validating a certificate, unable to access the specified LDAP directory.
- 428 – The specified key did not contain a private key.
- 429 – A failed attempt was made to load the specified Public-Key Cryptography Standards (PKCS) #11 shared library.
- 430 – The PKCS #11 driver failed to find the token specified by the caller.
- 431 – A PKCS #11 token is not present in the slot.
- 432 – The password/pin to access the PKCS #11 token is invalid.
- 433 – The SSL header received was not a properly SSLV2 formatted header.

- 501 – The buffer size is negative or zero.
- 502 – Used with non-blocking I/O. Refer to the non-blocking section for usage.
- 601 – SSLV3 is required for reset_cipher, and the connection uses SSLV2.
- 602 – An invalid ID was specified for the gsk_secure_soc_misc function call.
- 701 – The function call has an invalid ID. This may also be caused by specifying an environment handle when a handle for a SSL connection should be used.
- 702 – The attribute has a negative length, which is invalid.
- 703 – The enumeration value is invalid for the specified enumeration type.
- 704 – Invalid parameter list for replacing the SID cache routines.
- 705 – When setting a numeric attribute, the specified value is invalid for the specific attribute being set.
- 706 – Conflicting parameters have been set for additional certificate validation.

Chapter 4. Using GSK5IKM

The following key-management program is provided with the Tivoli Global Security Kit (GSKit):

- GSK5IKM - A user-friendly GUI for managing key database files, implemented as a Java applet.

Note: On the AIX operating systems, if you are prompted to set JAVA_HOME, you can set it to either the system-installed Java or the Java version included with the IBM Directory Server. If you use the IBM Directory Server version, you also need to set the LIBPATH environment variable as follows:

```
export LIBPATH=/usr/ldap/java/bin:/usr/ldap/java/bin/classic:$LIBPATH
```

Use this utility to create public-private key pairs and certificate requests, receive certificate requests into a key database file, and manage keys in a key database file.

The tasks you can perform with GSK5IKM include:

- Creating a key pair and requesting a certificate from a certificate authority
- Receiving a certificate into a key database file
- Managing keys and certificates
  - Changing a key database password
  - Showing information about a key
  - Deleting a key
  - Making a key the default key in the key database
  - Creating a key pair and certificate request for self-signing
  - Exporting a key
  - Importing a key into a key database
  - Designating a key as a trusted root
  - Removing trusted root key designation
  - Requesting a certificate for an existing key
- Migrating a keyring file to the key database format

Creating a key pair and requesting a certificate from a Certificate Authority

If your client application is connecting to an LDAP server that requires client and server authentication, then you need to create a public-private key pair and a certificate.

If your client application is connecting to an LDAP server that only requires server authentication, it is not necessary to create a public-private key pair and a certificate. It is sufficient to have a certificate in your client key database file that is marked as a trusted root. If the Certification Authority (CA) that issued the server's certificate is not already defined in your client key database, you need to request the CA's certificate from the CA, receive it into your key database, and mark it as trusted. See "Designating a key as a trusted root" on page 137.

Your client uses its private key to sign messages sent to servers. The server sends its public key to clients so that they can encrypt messages to the server, which the server decrypts with its private key.

To send its public key to a server, the client needs a certificate. The certificate contains the client's public key, the Distinguished Name associated with the client's certificate, the serial number of the certificate, and the expiration date of the certificate. A certificate is issued by a CA, which verifies the identity of the client.

The basic steps to create a certificate that is signed by a CA are:

1. Create a certificate request using GSK5IKM.
2. Submit the certificate request to the CA. This can be done using e-mail or an on-line submission from the CA's Web page.
3. Receive the response from the CA to an accessible location on the file system of your server.
4. Receive the certificate into your key database file.

Note: If you are obtaining a signed client certificate from a CA that is not in the default list of trusted CAs, you need to obtain the CA's certificate, receive it into your key database and mark it as trusted. This must be done before receiving your signed client certificate into the key database file.

To create a public-private key pair and request a certificate:

1. Start GSK5IKM Java utility by typing:
   ```
   GSK5IKM
   ```
2. Select **Key Database File**.
3. Select **New** (or **Open** if the key database already exists).
4. Specify key database file name and location. Type **OK**.

   Note: A key database is a file that the client or server uses to store one or more key pairs and certificates.
5. When prompted, supply password for the key database file. Click **OK**.
6. Select **Create**.
7. Select **New Certificate Request**.
8. Supply user-assigned label for key pair. The label identifies the key pair and certificate in the key database file.
9. If you are requesting a low-assurance client certificate, enter the common name. This must be unique and the full name of the user.
10. If you are requesting a high-assurance secure server certificate, then:
    - Enter the X.500 common name of the server. Usually this is the TCP/IP fully qualified host name, for example, www.ibm.com. For a VeriSign server certificate, it must be the fully qualified host name.
    - Enter the organization name. This is the name of your organization. For a VeriSign secure server certificate, if you already have an account with VeriSign, the name in this field must match the name on that account.
    - Enter the organizational unit name. This is an optional field.
    - Enter the locality/city where the server is located. This is an optional field.
    - Enter a three-character abbreviation of the state/province where the server is located.
    - Enter the postal code appropriate for the server's location.
    - Enter the two-character country code where the server is located.

11. Click **OK**.
12. A message identifying the name and location of the certificate request file is displayed. Click **OK**.
13. Send the certificate request to the CA.

    If this is a request for a VeriSign low assurance certificate or secure server certificate, you must e-mail the certificate request to VeriSign.

    You can mail the low assurance certificate request to VeriSign immediately. A secure server certificate request requires more documentation. To find out what VeriSign requires for a secure server certificate request, go to the following URL: http://www.verisign.com/ibm.
14. When you receive the certificate from the CA, use GSK5IKM to receive it into the key database where you stored the key pair. See "Receiving a certificate into a key database".

Note: Change the key database password frequently. If you specify an expiration date, you need to keep track of when you need to change the password. If the password expires before you change it, the key database is not usable until the password is changed.

Receiving a certificate into a key database

After receiving a response from your CA, you need to receive the certificate into a key database.

To receive a certificate into a key database:

1. Type `GSK5IKM` to start the Java utility.
2. Select **Key Database File**.
3. Select **Open**.
4. Specify key database file name and location. Type **OK**.
5. When prompted, supply password for the key database file, click **OK**.
6. Select **Create**.
7. Select **Personal Certificates** in the middle display window.
8. Click **Receive**.
9. Enter name and location of the certificate file that contains the signed certificate, as received from the CA. Click **OK**.

Changing a key database password

To change a key database password:

1. Type `GSK5IKM` to start the Java utility.
2. Select **Key Database File**.
3. Select **Open**.
4. Specify key database file name and location. Type **OK**.
5. When prompted, supply password for the key database file. Click **OK**.
6. Select **Key Database File**.
7. Select **Change Password**.
8. Enter *<New Password>*.
9. Confirm *<New Password>*.
10. Select and set optional password expiration time.

11. Select **Stash the password to a file?** if you want the password to be encrypted and stored on disk.
12. Click **OK**.
13. A message is displayed with the file name and location of the stash password file. Click **OK**.

Note: The password is important because it protects the private key. The private key is the only key that can sign documents or decrypt messages encrypted with the public key.

Showing information about a key

To show information about a key, such as its name, size or whether it is a trusted root:

1. Type `GSK5IKM` to start the Java utility.
2. Select **Key Database File**.
3. Select **Open**.
4. Specify key database file name and location. Type **OK**.
5. When prompted, supply password for the key database file. Click **OK**.
6. To see information about keys designated as Personal Certificates:
   - Select **Personal Certificates** at the top of the **Key database content** window.
   - Select a certificate.
   - Click **View/Edit** to display information about the selected key.
   - Click **OK** to return to the list of Personal Certificates.
7. To see information about keys that are designated as Signer Certificates:
   - Select **Signer Certificates** at the top of the **Key database content** window.
   - Select a certificate .
   - Click **View/Edit** to display information about the selected key.
   - Click **OK** to return to the list of Signer Certificates.

Deleting a key

To delete a key:

1. Type `GSK5IKM` to start the Java utility.
2. Select **Key Database File**.
3. Select **Open**.
4. Specify key database file name and location. Type **OK**.
5. When prompted, supply password for the key database file. Click **OK**.
6. Select the type of key you want to delete at the top of the **Key database content** window (Personal Certificates, Signer Certificates, or Personal Certificate Requests).
7. Select a certificate.
8. Click **Delete**.
9. Click **Yes** to confirm.

Making a key the default key in the key database

The default key must be the private key the server uses for its secure communications.

To make a key the default key in the key database:

1. Type `GSK5IKM` to start the Java utility.
2. Select **Key Database File**.
3. Select **Open**.
4. Specify key database file name and location. Type **OK**.
5. When prompted, supply password for the key database file. Click **OK**.
6. Select **Personal Certificates** at the top of the **Key database content** window.
7. Select the desired certificate.
8. Click **View/Edit**.
9. Select the **Set the certificates as the default** box. Click **OK**.

Creating a key pair and certificate request for self-signing

By definition, a secure server must have a public-private key pair and a certificate.

The server uses its private key to sign messages to clients. The server sends its public key to clients so they can encrypt messages to the server, which the server decrypts with its private key.

The server needs a certificate to send its public key to clients. The certificate contains the server's public key, the Distinguished Name associated with the server's certificate, the serial number of the certificate, and the expiration date of the certificate. A certificate is issued by a CA, who verifies the identity of the server.

You can request one of the following certificates:

- A low assurance certificate from VeriSign, best for non-commercial purposes, such as a beta test of your secure environment
- A server certificate to do commercial business on the Internet from VeriSign or some other CA
- A self-signed server certificate if you plan to act as your own CA for a private Web network

For information about using a CA such as VeriSign to sign the server certificate, see "Creating a key pair and requesting a certificate from a Certificate Authority" on page 131.

The basic steps to creating a self-signed certificate are:

1. Type `GSK5IKM` to start the Java utility.
2. Select **Key Database File**.
3. Select **New**, or **Open** if the key database already exists.
4. Specify key database file name and location. Type **OK**.

   Note: A key database is a file that the client or server uses to store one or more key pairs and certificates.
5. When prompted, supply password for the key database file. Click **OK**.
6. Click **New Self-signed**.

7. Supply the following:
   - User-assigned label for key pair. The label identifies the key pair and certificate in the key database file.
   - Select the desired certificate Version.
   - Select the desired Key Size.
   - Enter the X.500 common name of the server. Usually this is the TCP/IP fully qualified host name, for example, www.ibm.com.
   - Enter the organization name. This is the name of your organization.
   - Enter the organizational unit name. This is an optional field.
   - Enter the locality/city where the server is located. This is an optional field.
   - Enter a three-character abbreviation of the state/province where the server is located.
   - Enter the zipcode appropriate for the server's location.
   - Enter the two-character country code where the server is located.
   - Enter the Validity Period for the certificate.
8. Click **OK**.

.

Exporting a key

If you need to transfer a key pair or certificate to another computer, you can export the key pair from its key database to a file. On the other computer, you can import the key pair into a key ring.

To export a key from a key database:

1. Type `GSK5IKM` to start the Java utility.
2. Select **Key Database File**.
3. Select **Open**.
4. Specify key database file name and location. Type **OK**.
5. When prompted, supply password for the key database file. Click **OK**.
6. Select **Personal Certificates** at the top of the **Key database content** window.
7. Select the desired certificate.
8. Click **Export/Import**.
9. For **Action Type**, select **Export Key**.
10. Select the Key file type:
    - PKCS12 file
    - CMS Key database file
    - Keyring file (as used by mkkf)
    - SSLight key database class
11. Specify a file name.
12. Specify location.
13. Click **OK**.
14. Enter the required password for the file. Click **OK**.

Importing a key

To import a key into a key ring:

1. Type `GSK5IKM` to start the Java utility.
2. Select **Key Database File**.
3. Select **Open**.
4. Specify key database file name and location. Type **OK**.
5. When prompted, supply password for the key database file. Click **OK**.
6. Select **Personal Certificates** at the top of the **Key database content** window.
7. Select the desired certificate.
8. Click **Export/Import**.
9. For **Action Type**, select **Import Key**.
10. Select the desired Key file type.
11. Enter the file name and location.
12. Click **OK**.
13. Enter the required password for the source file. Click **OK**.

Designating a key as a trusted root

A trusted root key is the public key and associated Distinguished Name of a CA. The following trusted roots are automatically defined in each new key database:

- Integrion Certification Authority Root
- IBM World Registry Certification Authority
- Thawte Personal Premium CA
- Thawte Personal Freeemail CA
- Thawte Personal Basic CA
- Thawte Premium Server CA
- VeriSign Test CA Root Certificate
- RSA Secure Server Certification Authority
- VeriSign Class 1 Public Primary Certification Authority
- VeriSign Class 2 Public Primary Certification Authority
- VeriSign Class 3 Public Primary Certification Authority
- VeriSign Class 4 Public Primary Certification Authority

Note: Each of these trusted roots are initially set to be trusted roots by default.

To designate a key as a trusted root:

1. Type `GSK5IKM` to start the Java utility.
2. Select **Key Database File**.
3. Select **Open**.
4. Specify key database file name and location. Type **OK**.
5. When prompted, supply password for the key database file. Click **OK**.
6. Select **Signer Certificates** at the top of the **Key database content** window.
7. Select the desired certificate.
8. Click **View/Edit**.
9. Check the **Set the certificate as a trusted root** box, and click **OK**.
10. Select **Key Database File** and then select **Close**.

Removing a key as a trusted root

A trusted root key is the public key and associated Distinguished Name of a CA. The following trusted roots are automatically defined in each new key database:

- Integrion Certification Authority Root
- IBM World Registry Certification Authority
- Thawte Personal Premium CA
- Thawte Personal Freeemail CA
- Thawte Personal Basic CA
- Thawte Premium Server CA
- VeriSign Test CA Root Certificate
- RSA Secure Server Certification Authority
- VeriSign Class 1 Public Primary Certification Authority
- VeriSign Class 2 Public Primary Certification Authority
- VeriSign Class 3 Public Primary Certification Authority
- VeriSign Class 4 Public Primary Certification Authority

Note: Each of these trusted roots are initially set to be trusted roots by default.

To remove the trusted root status of a key:

1. Type `GSK5IKM` to start the Java utility.
2. Select **Key Database File**.
3. Select **Open**.
4. Specify key database file name and location. Type **OK**.
5. When prompted, supply password for the key database file. Click **OK**.
6. Select **Signer Certificates** at the top of the **Key database content** window.
7. Select the desired certificate.
8. Click **View/Edit**.
9. Uncheck the **Set the certificate as a trusted root** box. Click **OK**.
10. Select **Key Database File** and then select **Close**.

Requesting a certificate for an existing key

To create a certificate request for an existing key:

1. Type `GSK5IKM` to start the Java utility.
2. Select **Key Database File**.
3. Select **Open**.
4. Specify key database file name and location. Type **OK**.
5. When prompted, supply password for the key database file. Click **OK**.
6. Select **Personal Certificates** at the top of the **Key database content** window.
7. Select the desired certificate.
8. Click **Export/Import**.
9. For **Action Type**, select **Export Key**.
10. Select the desired Data Type:
    - Base-64-encoded ASCII data
    - Binary DER data
    - SSLight Key Database Class

11. Enter the certificate file name and location.
12. Click **OK**.
13. Select **Key Database File** and then select **Close**.

Send the certificate request to the CA.

If this is a request for a VeriSign low assurance certificate or secure server certificate, you must e-mail the certificate request to VeriSign.

You can mail the low assurance certificate request to VeriSign immediately. A secure server certificate request requires more documentation. To find out what VeriSign requires for a secure server certificate request, go to the following URL: http://www.verisign.com/ibm.

Migrating a keyring file to the key database format

The GSK5IKM program can be used to migrate an existing keyring file, as created with mkkf, to the format used by GSK5IKM.

To migrate a keyring file:

1. Type `GSK5IKM` to start the Java utility.
2. Select **Key Database File**.
3. Select **Open**.
4. Specify key database file name and location. Type **OK**.
5. When prompted, supply password for the keyring file. Click **OK**.
6. Select **Key Database File**.
7. Select **Save As...**.
8. Select **CMS key database file** as the Key database type.
9. Specify a file name.
10. Specify location.
11. Click **OK**.

Chapter 5. Event notification

The event notification function allows a server to notify a registered client that an entry in the directory tree has been changed, added or deleted. This notification is in the form of an unsolicited message.

Registration request

In order to register, the client must use a bound connection. To register a client use the supported client APIs for extended operations. An LDAP v3 extended operation request has the form:

```
ExtendedRequest ::= [APPLICATION 23] SEQUENCE {
        requestName      [0] LDAPOID,
        requestValue     [1] OCTET STRING OPTIONAL }
```

where the requestValue has the form:

```
requestValue = SEQUENCE {
        eventID          ENUMERATED {
                LDAP_CHANGE (0)},
        baseObject       LDAPDN,
        scope            ENUMERATED {
                baseObject              (0),
                singleLevel             (1),
                wholeSubtree            (2) },
        type    INTEGER OPTIONAL }
```

and where type has the form:

```
changeType ::= ENUMERATED {
                changeAdd               (1),
                changeDelete            (2),
                changeModify            (4),
                changeModDN             (8) }
```

Note: If the type field is not specified, it defaults to all changes.

An LDAP v3 extended operation response has the form:

```
ExtendedResponse ::= [APPLICATION 24] SEQUENCE {
        COMPONENTS OF LDAPResult,
        responseName     [10] LDAPOID OPTIONAL,
        response         [11] OCTET STRING OPTIONAL }
```

Registration response

If the registration is successful, the server returns the following message and a unique registration ID:

`LDAP_SUCCESS <registration ID>`

If the registration fails, the server returns one of the following:

`LDAP_UNWILLING_TO_PERFORM`

This error code is returned if:

- The event notification function is turned off in the server.
- The event ID requested by the client cannot be handled by the server.

- The client is unbound.

LDAP_NO_SUCH_OBJECT

This error code is returned if:

- The base DN supplied by the client does not exist or is not visible to the client.

LDAP_NOT_SUPPORTED

This error code is returned if:

- The change type supplied by the client cannot be handled by the server.

Usage

When an event occurs, the server sends a message to the client as an LDAP v3 unsolicited notification. The message ID is 0 and the message is in the form of an extended operation response. The responseName field is set to the registration OID. The response field contains the unique registration ID and a timestamp for when the event occurred. The time field is in Coordinated Universal Time (UTC) format.

Note: When a transaction occurs, the event notifications for the transaction steps cannot be sent until the entire transaction is completed.

Unregistering a client

Set the requestName field to the unregister request OID. In the requestValue field type the unique registration ID returned by the server from the registration request:

```
requestValue ::= OCTET STRING
```

If the registration is successfully removed, the LDAPResult field contains LDAP_SUCCESS and the response field contains the registration ID that was removed.

If the unregistration request was unsuccessful, NO_SUCH_OBJECT is returned.

Example

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


struct berval *create_reg(int id,char *base,int scope,int type){
  struct berval *ret;
  BerElement *ber;

  if((ber = ber_alloc_t(1)) == NULL){
    printf("ber_alloc_t failed\n");
    return NULL;
  }
  if(ber_printf(ber,"{esi",id,base,scope) == (-1)){
    printf("first ber_printf failed\n");
    return NULL;
  }
  if(type != (-1)){
    if(ber_printf(ber,"i",type) == (-1)){
      printf("type ber_printf failed\n");
      return NULL;
```

```
    }
  }
  if(ber_printf(ber,"}") == (-1)){
    printf("closing ber_printf failed\n");
    return NULL;
  }

  if(ber_flatten(ber,&ret) == (-1)){
    printf("ber_flatten failed\n");
    return NULL;
  }
  ber_free(ber,1);
  return ret;
}

int main(int argc,char **argv){
  LDAP *ld;
  char *oidreq = "1.3.18.0.2.12.1";
  char *oidres;
  struct berval *valres = NULL;
  struct berval *registration;
  int rc,version, port;
  LDAPMessage *res;
  BerElement *ber;
  char *regID;

  argc--; argv++;

  port = 389;
  if(argc > 0){
    if(argc > 1) sscanf(argv[1],"%d",&:port);
    ld = ldap_init(argv[0],port);
  }
  else
    ld = ldap_init("localhost",389);
  if(ld == NULL){
    printf("ldap_init failed\n");
    ldap_unbind(ld);
    return -1;
  }
  version = 3;
  ldap_set_option(ld,LDAP_OPT_PROTOCOL_VERSION,&version);

  if(ldap_simple_bind_s(ld,"cn=admin","secret") != LDAP_SUCCESS){
    printf("Couldn't bind\n");
    ldap_unbind(ld);
    return -1;
  }

  registration = create_reg(0,"o=ibm,c=us",2,15);
  rc = ldap_extended_operation_s(ld,oidreq,registration,NULL,NULL,
                                 &oidres,&valres);
  if(rc == LDAP_SUCCESS){
    if(valres != NULL){
      if((ber = ber_init2(valres)) == NULL)
        printf("ber_init2 failed\n");
      else{
        if(ber_scanf(ber,"a",&regID) == LBER_ERROR)
          printf("ber_scanf failed\n");
        printf("registration ID: %s\n",regID);
        ber_free(ber,1);
      }
    }
    else{
      printf("valres NULL\n");
    }
  }
```

```
  else{
    printf("extended operation failed 0x%x\n",rc);
  }

  // Wait for notifications
  printf("result: %d\n",ldap_result(ld,0,LDAP_MSG_ONE,NULL,&res));

  ldap_memfree(regID);
  ldap_unbind(ld);
  return 0;
}
```

Chapter 6. Limited transaction support

Transactions have four critical properties:

atomicity
The transaction must be performed completely. If any part of the transaction fails, the entire transaction is rolled back preserving the original state of the directory.

consistency
The transaction preserves the internal consistency of the database.

isolation
The transaction is serialized by a global lock so that it is performed independently of any other transactions.

durability
The results of a committed transaction are backed up in stable storage, usually a disk.

Usage

Transactions are limited to a single connection to a single IBM Directory server and are supported by the LDAP extended operations APIs. Only one transaction at a time can be running over the same connection. During the transaction, no nontransactional operations can be issued over the same connection.

A transaction consists of three parts:

- An extended request to start the transaction
- Update operations:
  - add
  - modify
  - modify rdn
  - delete

  Note: The current release does not support some operations, for example, bind, unbind, search, extended op, and so forth operations. Referral objects can be updated only with manageDsaIT control specified.
- An extended request to end the transaction

In order to start a transaction, the client must send an extended request in the form of:

```
ExtendedRequest ::= [APPLICATION 23] SEQUENCE {

requestName [0] LDAPOID,

requestValue [1] OCTET STRING OPTIONAL }
```

When the server receives the request, it generates a unique transaction ID. It then sends back an extended response in the form of:

```
ExtendedResponse ::= [APPLICATION 24]SEQUENCE{

COMPONENTS OF LDAPResult,
```

```
responseName [10] LDAPOID OPTIONAL,

response [11] OCTET STRING OPTIONAL }
```

The client submits subsequent update operations asynchronously with a control attached to all operations. The control contains the transaction ID returned in the StartTransaction response. The control has the form of:

```
Control ::= SEQUENCE {

controlType LDAPOID,

criticality BOOLEAN DEFAULT FALSE,

controlValue OCTET STRING OPTIONAL }
```

The server does not process update operations immediately. Instead, it saves the necessary information of operations in a queue.

The client sends an extended request to end the transaction that either commits or rolls back the transaction. The request has the same format as the start request. If the server receives the commit operation result, it uses a global writer lock to serialize the transaction. It then retrieves the set of update operations identified by the transaction ID from the queue and begins to perform these operations. If all operations succeed, the results are committed to the database and the server sends back the success return code.

As each operation is performed it generates a success return code unless an error occurs during the transaction, in which case an unsuccessful return code is returned for all the operations. If any operation fails, the server rolls back the transaction and sends back the error return code of the failed operation to the operation in the client that caused the failure. The EndTransaction operation also receives an unsuccessful return code if the transaction is not successful. For any subsequent update operations that still remain in the queue, an unsuccessful return code is generated. When the transaction times out, the connection is dropped and any subsequent operations receive an unsuccessful return code.

The server releases the global lock after the commit or the roll back is performed. The event notification and change log operations are performed only if the transaction has succeeded.

Example

The following example is an ldapmod.c example file, modified for limited transaction capability:

```
static char sccsid[] = "%Z%%M%  %I% %G% %W% %U%";
/*
 * COMPONENT_NAME: ldap.clients
 *
 * ABSTRACT: generic program to modify or add entries using LDAP with a transaction
 *
 * ORIGINS: 202,27
 *
 * (C) COPYRIGHT International Business Machines Corp. 2002
 * All Rights Reserved
 * Licensed Materials - Property of IBM
 *
 * US Government Users Restricted Rights - Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
```

```
 */

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

/* ldaptxmod.c - generic program to modify or add entries using LDAP
using a single transaction */

#include <ldap.h>

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/stat.h>

#if !defined( WIN32 )
#include <sys/file.h>
#include <fcntl.h>
#include <unistd.h>
#endif
#define LDAPMODIFY_REPLACE 1
#define LDAPMODIFY_ADD 2

#if defined( WIN32 )
#define strcasecmp stricmp
#endif

#define safe_realloc( ptr, size ) ( ptr == NULL ? malloc( size ) : \
 realloc( ptr, size ))

#define MAX_SUPPLIED_PW_LENGTH 256
#define LDAPMOD_MAXLINE 4096

/* Strings found in replog/LDIF entries (mostly lifted from slurpd/slurp.h) */
#define T_REPLICA_STR "replica"
#define T_DN_STR "dn"
#define T_CHANGENUMBER          "changenumber"
#define T_CHANGETYPESTR "changetype"
#define T_ADDCTSTR "add"
#define T_MODIFYCTSTR "modify"
#define T_DELETECTSTR "delete"
#define T_MODRDNCTSTR "modrdn"
#define T_MODOPADDSTR "add"
#define T_MODOPREPLACESTR "replace"
#define T_MODOPDELETESTR "delete"
#define T_MODSEPSTR "-"
#define T_NEWRDNSTR "newrdn"
#define T_DELETEOLDRDNSTR "deleteoldrdn"

extern char * str_getline(char**);
char * getPassword(void);
char * read_one_record(FILE *fp);

#if defined _WIN32
int getopt (int, char**, char*);
#endif
```

```
/* Global variables */
static LDAP     *ld         = NULL;  /* LDAP sesssion handle */
static FILE     *fp         = NULL;  /* input file handle */
static char *prog       = NULL;  /* program name */
static char *binddn     = NULL;  /* bind DN */
static char *passwd     = NULL;  /* bind password */
static char *ldaphost   = "localhost";  /* server host name */
static char     *mech       = NULL;  /* bind mechanism */
static char     *charset    = NULL;  /* character set for input */
static char     *keyfile    = NULL;  /* SSL key database file name*/
static char     *keyfile_pw = NULL;  /* SSL key database password */
static char     *cert_label = NULL;  /* client certificate label */
static int      hoplimit  = 10;      /* limit for referral chasing */
static int ldapport  = LDAP_PORT; /* server port number */
static int doit      = 1;       /* 0 to make believe */
static int verbose   = 0;       /* 1 for more trace messages */
static int contoper  = 0;       /* 1 to continue after errors */
static int force     = 0;
static int valsfromfiles = 0;
static int operation   = LDAPMODIFY_REPLACE;
static int referrals   = LDAP_OPT_ON;
static int ldapversion = LDAP_VERSION3;
static int DebugLevel  = 0;          /* 1 to activate library traces */
static int ssl         = 0;          /* 1 to use SSL */
static int  manageDsa  = LDAP_FALSE; /* LDAP_TRUE to modify referral objects */

static LDAPControl manageDsaIT = {
  "2.16.840.1.113730.3.4.2", /* OID */
  { 0, NULL },               /* no value */
  LDAP_OPT_ON                /* critical */
};

/* NULL terminated array of server controls*/
static LDAPControl *Server_Controls[3] = {NULL, NULL, NULL};

static int Num_Operations = 0;  /* count of times one must go to
        ldap_result to check result codes */
static int Message_ID = 0;      /* message ID returned by async
        ldap operation, currently not tracked*/
static int abort_flag = 0;      /* abort transaction flag set by
        -A parameter */

/* Implement getopt() for Windows to parse command line arguments. */
#if defined(_WIN32)
char *optarg = NULL;
int   optind = 1;
int   optopt = 0;
#define EMSG ""

int getopt(int argc, char **argv, char *ostr) {
  static char *place = EMSG;
  register char *oli;

  if (!*place) {
    if (optind >= argc || *(place = argv[optind]) != '-' || !*++place) {
      return EOF;
    }
    if (*place == '-') {
      ++optind;
      return EOF;
    }
  }
  if ((optopt = (int)*place++) == (int)':' || !(oli = strchr(ostr, optopt))) {
    if (!*place) {
      ++optind;
    }
```

```
    fprintf(stderr, "%s: %s: %c\n", "getopt", "illegal option", optopt);
    return ( '?' );
  }
  if (***oli != ':') {
    optarg = NULL;
    if (!*place)
      ++optind;
  } else {
    if (*place) {
      optarg = place;
    } else if (argc <= ++optind) {
      place = EMSG;
      fprintf(stderr, "%s: %s: %c\n", "getopt", "option requires an argument", optopt);
      return 0;
    } else {
      optarg = argv[optind];
    }
    place = EMSG;
    ++optind;
  }
  return optopt;
}
#endif

/* Display usage statement and exit. */
void usage()
{
  fprintf(stderr, "\nSends modify or add requests to an LDAP server.\n");
  fprintf(stderr, "usage:\n");
  fprintf(stderr, "    %s [options] [-f file]\n", prog);
  fprintf(stderr, "where:\n");
  fprintf(stderr, "    file: name of input file\n");
  fprintf(stderr, "note:\n");
  fprintf(stderr, "    standard input is used if file is not specified\n");
  fprintf(stderr, "options:\n" );
  fprintf(stderr, "    -h host      LDAP server host name\n");
  fprintf(stderr, "    -p port      LDAP server port number\n");
  fprintf(stderr, "    -D dn        bind DN\n");
  fprintf(stderr, "    -w password  bind password or '?' for non-echoed prompt\n");
  fprintf(stderr, "    -Z           use a secure ldap connection (SSL)\n");
  fprintf(stderr, "    -K keyfile   file to use for keys\n");
  fprintf(stderr, "    -P key_pw    keyfile password\n");
  fprintf(stderr, "    -N key_name  private key name to use in keyfile\n");
  fprintf(stderr, "    -R           do not chase referrals\n");
  fprintf(stderr, "    -M           Manage referral objects as normal entries.\n");
  fprintf(stderr, "    -m mechanism perform SASL bind with the given mechanism\n");
  fprintf(stderr, "    -O maxhops   maximum number of referrals to follow in a sequence\n");
  fprintf(stderr, "    -V version   LDAP protocol version (2 or 3; only 3 is supported)\n");
  fprintf(stderr, "    -C charset   character set name to use, as registered with IANA\n");
  fprintf(stderr, "    -a           force add operation as default\n");
  fprintf(stderr, "    -r           force replace operation as default\n");
  fprintf(stderr, "    -b           support binary values from files (old style paths)\n");
  fprintf(stderr, "    -c           continuous operation; do not stop processing on error\n");
  fprintf(stderr, "    -n           show what would be done but don't actually do it\n");
  fprintf(stderr, "    -v           verbose mode\n");
  fprintf(stderr, "    -A           set transaction abort flag\n");
  fprintf(stderr, "    -d level     set debug level in LDAP library\n");
  exit(1);
}

/* Parse command line arguments. */
void parse_arguments(int argc, char **argv) {
  int i = 0;
  int port = 0;
  char *optpattern = "FaAbcRMZnrv?h:V:p:D:w:d:f:K:P:N:C:O:m:";
#ifndef _WIN32
  extern char *optarg;
```

```
  extern int optind;
#endif

  fp = stdin;
  while ((i = getopt(argc, argv, optpattern)) != EOF) {
    switch ( i ) {
    case 'V':
      ldapversion = atoi(optarg);
      if (ldapversion != LDAP_VERSION3) {
fprintf(stderr, "Unsupported version level supplied.\n");
usage();
      }
      break;
    case 'A':       /* force all changes records to be used */
      abort_flag = 1;
      break;
    case 'a':
      operation = LDAPMODIFY_ADD;
      break;
    case 'b': /* read values from files (for binary attributes)*/
      valsfromfiles = 1;
      break;
    case 'c': /* continuous operation*/
      contoper = 1;
      break;
    case 'F': /* force all changes records to be used*/
      force = 1;
      break;
    case 'h': /* ldap host*/
      ldaphost = strdup( optarg );
      break;
    case 'D': /* bind DN */
      binddn = strdup( optarg );
      break;
    case 'w': /* password*/
      if (optarg && optarg[0] == '?') {
passwd = getPassword();
      } else
if (!(passwd = strdup( optarg )))
  perror("password");
      break;
    case 'd':
      DebugLevel = atoi(optarg);
      break;
    case 'f': /* read from file */
      if ((optarg[0] == '-') && (optarg[1] == '\0'))
fp = stdin;
      else if ((fp = fopen( optarg, "r" )) == NULL) {
perror( optarg );
exit( 1 );
      }
      break;
    case 'p':
      ldapport = atoi( optarg );
      port = 1;
      break;
    case 'n': /* print adds, don't actually do them*/
      doit = 0;
      break;
    case 'r': /* default is to replace rather than add values*/
      operation = LDAPMODIFY_REPLACE;
      break;
    case 'R':  /* don't automatically chase referrals*/
      referrals = LDAP_OPT_OFF;
      break;
    case 'M':   /* manage referral objects as normal entries */
      manageDsa = LDAP_TRUE;
```

```
      break;
    case 'O':   /* set maximum referral hop count  */
      hoplimit = atoi( optarg );
      break;
    case 'm':   /* use SASL bind mechanism  */
      if (!(mech = strdup ( optarg )))
perror("mech");
      break;
    case 'v':   /* verbose mode */
      verbose++;
      break;
    case 'K':
      keyfile = strdup( optarg );
      break;
    case 'P':
      keyfile_pw = strdup( optarg );
      break;
    case 'N':
      cert_label = strdup( optarg );
      break;
    case 'Z':
      ssl = 1;
      break;
    case 'C':
      charset = strdup(optarg);
      break;
    case '?':
    default:
      usage();
    }
  }

  if (argc - optind !=  0)
    usage();

  /* Use default SSL port if none specified*/
  if (( port == 0 ) && ( ssl ))
    ldapport = LDAPS_PORT;

  if ( ! DebugLevel ) {
    char *debug_ptr = NULL;

    if ( ( debug_ptr = getenv ( "LDAP_DEBUG" ) ) )
      DebugLevel = atoi ( debug_ptr );
  }
}

/* Get a password from the user but don't display it. */
char* getPassword( void ) {
  char supplied_password[ MAX_SUPPLIED_PW_LENGTH + 1 ]; /* Buffer for password */

#ifdef _WIN32
  char in  = '\0';                /* Input character */
  int  len = 0;                   /* Length of password */
#else
  struct termios echo_control;
  struct termios save_control;

  int fd = 0;                 /* File descriptor */
  int attrSet = 0;            /* Checked later for reset */

  /* Get the file descriptor associated with stdin. */
  fd = fileno( stdin );

  if (tcgetattr( fd, &echo_control ) != -1) {
    save_control = echo_control;
    echo_control.c_lflag &= ~( ECHO | ECHONL );
```

```
    if (tcsetattr( fd, TCSANOW, &echo_control ) == -1) {
      fprintf(stderr, "Internal error setting terminal attribute.\n");
      exit( errno );
    }

    attrSet = 1;
  }
#endif

  /* Prompt for a password. */
  fputs( "Enter password ==> ", stdout );
  fflush( stdout );

#ifdef _WIN32
  /* Windows 9x/NT will always read from the console, i.e.,
     piped or redirected input will be ignored. */
  while ( in != '\r' && len <= MAX_SUPPLIED_PW_LENGTH ) {
    in = _getch();

    if (in != '\r') {
      supplied_password[len] = in;
      len++;
    } else {
      supplied_password[len] = '\0';
    }
  }
#else
  /* Get the password from stdin. */
  fgets( supplied_password, MAX_SUPPLIED_PW_LENGTH, stdin );

  /* Remove the newline at the end. */
  supplied_password[strlen( supplied_password ) - 1] = '\0';

#endif

#ifndef _WIN32
  /* Reset the terminal. */
  if (attrSet && tcsetattr( fd, TCSANOW, &save_control ) == -1) {
    fprintf(stderr, "Unable to reset the display.\n");
  }
#endif
  fprintf( stdout, "\n" );

  return ( supplied_password == NULL )? supplied_password : strdup( supplied_password );
}

/* Rebind callback function. */
int rebindproc(LDAP *ld, char **dnp, char **pwp, int *methodp, int freeit) {
  if ( !freeit ) {
    *methodp = LDAP_AUTH_SIMPLE;
    if ( binddn != NULL ) {
      *dnp = strdup( binddn );
      *pwp = strdup ( passwd );
    } else {
      *dnp = NULL;
      *pwp = NULL;
    }
  } else {
    free ( *dnp );
    free ( *pwp );
  }
  return LDAP_SUCCESS;
}

/* Connect and bind to server. */
void connect_to_server() {
```

```
  int failureReasonCode, rc, authmethod;
  struct berval ber;
  struct berval *server_creds;

  /* call ldap_ssl_client_init if V3 and SSL */
  if (ssl && (ldapversion == LDAP_VERSION3)) {
    if ( keyfile == NULL ) {
      keyfile = getenv("SSL_KEYRING");
      if (keyfile != NULL) {
keyfile = strdup(keyfile);
      }
    }

    if (verbose)
      printf( "ldap_ssl_client_init( %s, %s, 0, &failureReasonCode )\n",
      ((keyfile) ? keyfile : "NULL"),
      ((keyfile_pw) ? keyfile_pw : "NULL"));
#ifdef LDAP_SSL_MAX
    rc = ibm_set_unrestricted_cipher_support();
    if (rc != 0) {
      fprintf( stderr, "Warning: ibm_gsk_set_unrestricted_cipher_support failed!
        rc == %d\n", rc );
    }
#endif

    rc = ldap_ssl_client_init( keyfile, keyfile_pw, 0, &failureReasonCode );
    if (rc != LDAP_SUCCESS) {
      fprintf( stderr,
       "ldap_ssl_client_init failed! rc == %d, failureReasonCode == %d\n",
       rc, failureReasonCode );
      exit( 1 );
    }
  }
  /* Open connection to server */
  if (ldapversion == LDAP_VERSION3) {
    if (ssl) {
      if (verbose)
printf("ldap_ssl_init( %s, %d, %s )\n", ldaphost, ldapport,
       ((cert_label) ? cert_label : "NULL"));
      ld = ldap_ssl_init( ldaphost, ldapport, cert_label );
      if (ld == NULL) {
fprintf( stderr, "ldap_ssl_init failed\n" );
perror( ldaphost );
exit( 1 );
      }
    } else {
      if (verbose)
printf("ldap_init(%s, %d) \n", ldaphost, ldapport);
      if ((ld = ldap_init(ldaphost, ldapport)) == NULL) {
perror(ldaphost);
exit(1);
      }
    }
  }

  /* Set options */
  ldap_set_option (ld, LDAP_OPT_PROTOCOL_VERSION, (void * )&ldapversion);

  if (ldapversion == LDAP_VERSION3) {
    ldap_set_option (ld, LDAP_OPT_DEBUG, (void * )&DebugLevel);
    ldap_set_option( ld, LDAP_OPT_REFHOPLIMIT, (void *)&hoplimit);
  }
  ldap_set_option (ld, LDAP_OPT_REFERRALS, (void * )referrals);
  if (binddn != NULL)
    ldap_set_rebind_proc( ld, (LDAPRebindProc)rebindproc );
  if (charset != NULL) {
    if (ldap_set_iconv_local_charset(charset) != LDAP_SUCCESS) {
```

```
      fprintf(stderr, "unsupported charset %s\n", charset);
      exit(0);
    }
    ldap_set_option(ld, LDAP_OPT_UTF8_IO, (void *)LDAP_UTF8_XLATE_ON);
  }

  /* Bind to server */
  if (ldapversion == LDAP_VERSION3) {
    if ( ! mech ) /* Use simple bind */ {
      rc = ldap_simple_bind_s(ld, binddn, passwd);
      if ( rc != LDAP_SUCCESS ) {
ldap_perror( ld, "ldap_simple_bind" );
/* LDAP_OPT_EXT_ERROR only valuable for ssl communication.
   In this example, for LDAP v3, the bind is the first
   instance in which communication actually flows to the
   server.  So, if there is an ssl configuration error or
   other ssl problem, this will be the first instance where
   it will be detected. */
if (ssl) {
  ldap_get_option( ld, LDAP_OPT_EXT_ERROR, &failureReasonCode);
  fprintf( stderr, "Attempted communication over SSL.\n");
  fprintf( stderr, "  The extended error is %d.\n", failureReasonCode);
}
exit( rc );
      }
    } else /* Presence of mechanism means SASL bind */ {
      /* Special case for mech="EXTERNAL".  Unconditionally set bind DN
 and credentials to NULL.  This option should be used in tandem
 with SSL and client authentication.  For other SASL mechanisms,
 use the specified bind DN and credentials. */
      if (strcmp(mech, LDAP_MECHANISM_EXTERNAL) == 0) {
rc = ldap_sasl_bind_s (ld, NULL, mech, NULL, NULL, NULL, &server_creds);
if (rc != LDAP_SUCCESS ) {
  ldap_perror ( ld, "ldap_sasl_bind_s" );
  exit( rc );
}
      } else {
if (strcmp(mech, LDAP_MECHANISM_GSSAPI) == 0) {
  rc = ldap_sasl_bind_s (ld, NULL, mech, NULL, NULL, NULL, &server_creds);
  if (rc != LDAP_SUCCESS ) {
    ldap_perror ( ld, "ldap_sasl_bind_s" );
    exit( rc );
  }
} else /* other SASL mechanisms */ {
  ber.bv_len = strlen ( passwd );
  ber.bv_val = passwd;
  rc = ldap_sasl_bind_s (ld, binddn, mech, &ber, NULL, NULL, &server_creds);
  if (rc != LDAP_SUCCESS ) {
    ldap_perror ( ld, "ldap_sasl_bind_s" );
    exit( rc );
  }
}
      }
    }
  }
}

/* Read a record from the file. */
char * read_one_record(FILE *fp)
{
  int len = 0;
  int lcur = 0;
  int   lmax = 0;
  char line[LDAPMOD_MAXLINE];
  char temp[LDAPMOD_MAXLINE];
  char *buf = NULL;
  /* Reads in and changes to ldif form */
```

```
  while (( fgets( line, sizeof(line), fp ) != NULL )) {
    if (!(strncmp(line,"changenumber",10)))
      {do
fgets(line,sizeof(line),fp);
      while(strncmp(line,"targetdn",8)); /*changes the = to : for parse*/
      line[8]=':';}
    if (!(strncmp(line,"changetype",9)))
      line[10]=':';
    if (!(strncmp(line,"changetype:delete",16)))
      (fgets(temp,sizeof(line),fp)); /*gets rid of the changetime line after a delete.*/
    if (!(strncmp(line,"changetime",9)))
      {fgets(line,sizeof(line),fp);
      if (!(strncmp(line,"newrdn",6)))
line[6]=':';
      else
line[7]=':';
      }
    if (!(strncmp(line,"deleteoldrdn",12)))
      line[12]=':';
    if ( *line != '\n' ) {
      len = strlen( line );
      if ( lcur + len + 1 > lmax ) {
lmax = LDAPMOD_MAXLINE
  *(( lcur + len + 1 ) / LDAPMOD_MAXLINE + 1 );
if (( buf = (char *)safe_realloc( buf, lmax )) == NULL ) {
  perror( "safe_realloc" );
  exit( 1 );
}
      }
      strcpy( buf + lcur, line );
      lcur += len;
    }
    else {
      if ( buf == NULL )
continue; /* 1st line keep going */
      else
break;
    }
  }

  return buf;
}

/* Read binary data from a file. */
int fromfile(char *path, struct berval *bv) {
  FILE *fp  = NULL;
  long  rlen = 0;
  int  eof  = 0;

  /* "r" changed to "rb", defect 39803. */
  if (( fp = fopen( path, "rb" )) == NULL ) {
    perror( path );
    return -1;
  }

  if ( fseek( fp, 0L, SEEK_END ) != 0 ) {
    perror( path );
    fclose( fp );
    return -1;
  }

  bv->bv_len = ftell( fp );

  if (( bv->bv_val = (char *)malloc( bv->bv_len )) == NULL ) {
    perror( "malloc" );
    fclose( fp );
    return -1;
```

```
  }

  if ( fseek( fp, 0L, SEEK_SET ) != 0 ) {
    perror( path );
    fclose( fp );
    return -1;
  }

  rlen = fread( bv->bv_val, 1, bv->bv_len, fp );
  eof = feof( fp );
  fclose( fp );

  if ( rlen != (bv->bv_len) ) {
    perror( path );
    return -1;
  }

  return bv->bv_len;
}

/* Read binary data from a file specified with a URL. */
int fromfile_url(char *value, struct berval *bv) {
  char *file = NULL;
  char *src  = NULL;
  char *dst  = NULL;

  if (strncmp(value, "file:///", 8))
    return -1;

  /* unescape characters */
  for (dst = src = &value[8]; (*src != '\0'); ++dst) {
    *dst = *src;
    if (*src++ != '%')
      continue;
    if ((*src >= '0') && (*src <= '9'))
      *dst = (*src++ - '0') << 4;
    else if ((*src >= 'a') && (*src <= 'f'))
      *dst = (*src++ - 'a' + 10) << 4;
    else if ((*src >= 'A') && (*src <= 'F'))
      *dst = (*src++ - 'A' + 10) << 4;
    else
      return -1;
    if ((*src >= '0') && (*src <= '9'))
      *dst += (*src++ - '0');
    else if ((*src >= 'a') && (*src <= 'f'))
      *dst += (*src++ - 'a' + 10);
    else if ((*src >= 'A') && (*src <= 'F'))
      *dst += (*src++ - 'A'+ 10);
    else
      return -1;
  }
  *dst = '\0';

  /* On WIN32 platforms the URL must begin with a drive letter.
     On UNIX platforms the initial '/' is kept to indicate absolute
     file path.
  */
#ifdef _WIN32
  file = value + 8;
#else
  file = value + 7;
#endif
  return fromfile(file, bv);
}

/* Add operation to the modify structure. */
void addmodifyop(LDAPMod ***pmodsp, int modop, char *attr,
```

```
 char *value, int vlen, int isURL, int isBase64)
{
  LDAPMod **pmods = NULL;
  int i = 0;
  int j = 0;
  struct berval *bvp = NULL;

  /* Data can be treated as binary (wire ready) if one of the
     following applies:
     1) it was base64 encoded
     2) charset is not defined
     3) read from an external file
  */
  if (isBase64 ||
      (charset == NULL) ||
      isURL ||
      ((value != NULL) && valsfromfiles && (*value == '/'))) {
    modop |= LDAP_MOD_BVALUES;
  }

  i = 0;
  pmods = *pmodsp;
  if ( pmods != NULL ) {
    for (; pmods[ i ] != NULL; ++i ) {
      if ( strcasecmp( pmods[ i ]->mod_type, attr ) == 0 &&
   pmods[ i ]->mod_op == modop ) {
break;
      }
    }
  }

  if ( pmods == NULL || pmods[ i ] == NULL ) {
    if (( pmods = (LDAPMod * *)safe_realloc( pmods, (i + 2) *
     sizeof( LDAPMod * ))) == NULL ) {
      perror( "safe_realloc" );
      exit( 1 );
    }
    *pmodsp = pmods;
    pmods[ i + 1 ] = NULL;
    if (( pmods[ i ] = (LDAPMod * )calloc( 1, sizeof( LDAPMod ))) == NULL ) {
      perror( "calloc" );
      exit( 1 );
    }
    pmods[ i ]->mod_op = modop;
    if (( pmods[ i ]->mod_type = strdup( attr )) == NULL ) {
      perror( "strdup" );
      exit( 1 );
    }
  }

  if ( value != NULL ) {
    if (modop & LDAP_MOD_BVALUES) {
      j = 0;
      if ( pmods[ i ]->mod_bvalues != NULL ) {
for (; pmods[ i ]->mod_bvalues[ j ] != NULL; ++j ) {
  ;
}
      }
      if (( pmods[ i ]->mod_bvalues =
    (struct berval **)safe_realloc( pmods[ i ]->mod_bvalues,
    (j + 2) * sizeof( struct berval *))) == NULL ) {
perror( "safe_realloc" );
exit( 1 );
      }

      pmods[ i ]->mod_bvalues[ j + 1 ] = NULL;
      if (( bvp = (struct berval *)malloc( sizeof( struct berval )))
```

```
  == NULL ) {
perror( "malloc" );
exit( 1 );
      }
      pmods[ i ]->mod_bvalues[ j ] = bvp;

      /* get value from file */
      if ( valsfromfiles && *value == '/' ) {
if (fromfile( value, bvp ) < 0 )
  exit(1);
      } else if (isURL) {
if (fromfile_url(value, bvp) < 0)
  exit(1);
      } else {
bvp->bv_len = vlen;
if (( bvp->bv_val = (char *)malloc( vlen + 1 )) == NULL ) {
  perror( "malloc" );
  exit( 1 );
}
memmove( bvp->bv_val, value, vlen );
bvp->bv_val[ vlen ] = '\0';
      }
    } else {
      j = 0;
      if ( pmods[ i ]->mod_values != NULL ) {
for ( ; pmods[ i ]->mod_values[ j ] != NULL; ++j ) {
  ;
}
      }
      if (( pmods[ i ]->mod_values =
    (char **)safe_realloc( pmods[ i ]->mod_values,
   (j + 2) * sizeof( char *))) == NULL ) {
perror( "safe_realloc" );
exit( 1 );
      }
      pmods[ i ]->mod_values[ j + 1 ] = NULL;
      if (( pmods[ i ]->mod_values[ j ] = strdup( value )) == NULL) {
perror( "strdup" );
exit( 1 );
      }
    }
  }
}

/* Delete record */
int dodelete( char *dn ) {
  int rc = 0;

  printf( "%sdeleting entry %s\n", (!doit) ? "!" : "", dn );
  if (!doit)
    return LDAP_SUCCESS;

  rc = ldap_delete_ext( ld, dn,
Server_Controls,
NULL, &Message_ID);
  if ( rc != LDAP_SUCCESS )
    ldap_perror( ld, "ldap_delete" );
  else
    printf( "delete complete\n" );

  putchar('\n');
  /* Increment results to check after end transaction. */
  Num_Operations++;
  return rc;
}

/* Copy or move an entry. */
```

```
int domodrdn( char *dn, char *newrdn, int deleteoldrdn ) {
  int rc = 0;

  printf( "%s%s %s to %s\n", ((!doit) ? "!" : ""),
  ((deleteoldrdn) ? "moving" : "copying"), dn, newrdn);
  if (!doit)
    return LDAP_SUCCESS;

  rc = ldap_rename( ld, dn, newrdn, NULL, deleteoldrdn,
    Server_Controls , NULL,
    &Message_ID );
  if ( rc != LDAP_SUCCESS )
    ldap_perror( ld, "ldap_rename" );
  else
    printf( "rename operation complete\n" );
  putchar('\n');

  /* Increment the count of results to check after end transaction is sent */
  Num_Operations++;
  return rc;
}

/* Print a binary value. If charset is not specified then check to
   see if string is printable anyway. */
void print_binary(struct berval *bval) {
  int i = 0;
  int binary = 0;
  printf( "\tBINARY (%ld bytes) ", bval->bv_len);
  if (charset == NULL) {
    binary = 0;
    for (i = 0; (i < (bval->bv_len)) && (!binary); ++i)
      if (!isprint(bval->bv_val[i]))
binary = 1;
    if (!binary)
      for (i = 0; (i < (bval->bv_len)); ++i)
putchar(bval->bv_val[i]);
  }
  putchar('\n');
}

/* Modify or add an entry. */
int domodify( char *dn, LDAPMod **pmods, int newentry ) {
  int i, j, op, rc;
  struct berval *bvp;

  if ( pmods == NULL ) {
    fprintf( stderr, "%s: no attributes to change or add (entry %s)\n",
     prog, dn );
    return LDAP_PARAM_ERROR;
  }

  if ( verbose ) {
    for ( i = 0; pmods[ i ] != NULL; ++i ) {
      op = pmods[ i ]->mod_op & ~LDAP_MOD_BVALUES;
      printf( "%s %s:\n", op == LDAP_MOD_REPLACE ?
      "replace" : op == LDAP_MOD_ADD ?
      "add" : "delete", pmods[ i ]->mod_type );
      if (pmods[i]->mod_op & LDAP_MOD_BVALUES) {
if (pmods[ i ]->mod_bvalues != NULL) {
  for (j = 0; pmods[i]->mod_bvalues[j] != NULL; ++j)
    print_binary(pmods[i]->mod_bvalues[j]);
}
      } else {
if (pmods[i]->mod_values != NULL) {
  for (j = 0; pmods[i]->mod_values[j] != NULL; ++j)
    printf("\t%s\n", pmods[i]->mod_values[j]);
}
```

```
      }
    }
  }

  if ( newentry )
    printf( "%sadding new entry %s as a transaction\n", (!doit) ? "!" : "", dn );
  else
    printf( "%smodifying entry %s as a transaction\n", (!doit) ? "!" : "", dn );
  if (!doit)
    return LDAP_SUCCESS;

  if ( newentry ) {
    rc = ldap_add_ext( ld, dn, pmods,
       Server_Controls, NULL,
       &Message_ID);
  } else {
    rc = ldap_modify_ext( ld, dn, pmods,
  Server_Controls, NULL,
  &Message_ID );
  }
  if ( rc != LDAP_SUCCESS ) {
    ldap_perror( ld, newentry ? "ldap_add" : "ldap_modify" );
  } else if ( verbose ) {
    printf( "%s operation complete\n", newentry ? "add" : "modify" );
  }
  putchar( '\n' );

  /* Increment the count of results to check after end transaction is sent */
  Num_Operations++;
  return rc;
}

/* Process an ldif record. */
int process_ldif_rec(char *rbuf) {
  char *line     = NULL;
  char *dn       = NULL;
  char *type     = NULL;
  char *value    = NULL;
  char *newrdn   = NULL;
  char *p        = NULL;
  int is_url   = 0;
  int   is_b64   = 0;
  int rc       = 0;
  int   linenum  = 0;
  int   vlen     = 0;
  int   modop    = 0;
  int   replicaport   = 0;
  int expect_modop  = 0;
  int   expect_sep    = 0;
  int   expect_ct     = 0;
  int   expect_newrdn = 0;
  int expect_deleteoldrdn = 0;
  int   deleteoldrdn  = 1;
  int saw_replica   = 0;
  int use_record      = force;
  int new_entry = (operation == LDAPMODIFY_ADD);
  int delete_entry = 0;
  int got_all = 0;
  LDAPMod **pmods = NULL;
  int version = 0;
  int str_rc  = 0;

  while ( rc == 0 && ( line = str_getline( &rbuf )) != NULL ) {
    ++linenum;

    /* Is this a separator line ("-")? */
    if ( expect_sep && strcasecmp( line, T_MODSEPSTR ) == 0 ) {
```

```
      /* If modifier has not been added yet then go ahead and add
 it. The can happen on sequences where there are no
 attribute values, such as:
 DELETE: title
 -
      */
      if (value != NULL)
addmodifyop(&pmods, modop, value, NULL, 0, 0, 0);
      value = NULL;
      expect_sep = 0;
      expect_modop = 1;
      continue;
    }

    str_rc = str_parse_line_v_or_bv(line, &type, &value, &vlen, 1, &is_url, &is_b64);
    if ((strncmp(type,"changes",7))==0)
      {str_parse_line_v_or_bv(value, &type, &value, &vlen, 1, &is_url, &is_b64);}
    if ((linenum == 1) && (strcmp(type, "version") == 0)) {
      version = atoi(value);
      continue;
    }
    if ((linenum == 2) && (version == 1) &&
(strcmp(type, "charset") == 0)) {
      if (charset != NULL)
free(charset);
      charset = strdup(value);
      if ((rc = ldap_set_iconv_local_charset(charset)) != LDAP_SUCCESS) {
fprintf(stderr, "unsupported charset %s\n", charset);
break;
      }
      ldap_set_option(ld, LDAP_OPT_UTF8_IO, (void *)LDAP_UTF8_XLATE_ON);
      continue;
    }

    if ( dn == NULL ) {
      if ( !use_record && strcasecmp( type, T_REPLICA_STR ) == 0 ) {
++saw_replica;
if (( p = strchr( value, ':' )) == NULL ) {
  replicaport = LDAP_PORT;
} else {
  *p++ = '\0';
  replicaport = atoi( p );
}
if ( strcasecmp( value, ldaphost ) == 0 &&
     replicaport == ldapport ) {
  use_record = 1;
}
      } else if ( strcasecmp( type, T_DN_STR ) == 0 ) {
if (( dn = strdup( value )) == NULL ) {
  perror( "strdup" );
  exit( 1 );
}
expect_ct = 1;
      }
      continue; /* skip all lines until we see "dn:" */
    }

    if ( expect_ct ) {
      expect_ct = 0;
      if ( !use_record && saw_replica ) {
printf( "%s: skipping change record for entry: %s\n\t(LDAP host/port does
        not match replica: lines)\n", prog, dn );
free( dn );
return 0;
      }

      /* this is an ldif-change-record */
```

```
      if ( strcasecmp( type, T_CHANGETYPESTR ) == 0 ) {
if ( strcasecmp( value, T_MODIFYCTSTR ) == 0 ) {
  new_entry = 0;
  expect_modop = 1;
} else if ( strcasecmp( value, T_ADDCTSTR ) == 0 ) {
  modop = LDAP_MOD_ADD;
  new_entry = 1;
} else if ( strcasecmp( value, T_MODRDNCTSTR ) == 0 ) {
  expect_newrdn = 1;
} else if ( strcasecmp( value, T_DELETECTSTR ) == 0 ) {
  got_all = delete_entry = 1;
} else {
  fprintf( stderr,
   "%s:  unknown %s \"%s\" (line %d of entry: %s)\n",
   prog, T_CHANGETYPESTR, value, linenum, dn );
  rc = LDAP_PARAM_ERROR;
}
continue;

/* this is an ldif-attrval-record */
      } else {
if (operation == LDAPMODIFY_ADD) {
  new_entry = 1;
  modop = LDAP_MOD_ADD;
} else
  modop = LDAP_MOD_REPLACE;
      }
    }

    if (expect_modop) {
      expect_modop = 0;
      expect_sep = 1;
      if ( strcasecmp( type, T_MODOPADDSTR ) == 0 ) {
modop = LDAP_MOD_ADD;
continue;
      } else if ( strcasecmp( type, T_MODOPREPLACESTR ) == 0 ) {
modop = LDAP_MOD_REPLACE;
continue;
      } else if ( strcasecmp( type, T_MODOPDELETESTR ) == 0 ) {
modop = LDAP_MOD_DELETE;
continue;
      } else {
fprintf(stderr,
"%s: unknown mod_spec \"%s\" (line %d of entry: %s)\n",
prog, type, linenum, dn);
rc = LDAP_PARAM_ERROR;
continue;
      }
    }

    if ( expect_newrdn ) {
      if ( strcasecmp( type, T_NEWRDNSTR ) == 0 ) {
if (( newrdn = strdup( value )) == NULL ) {
  perror( "strdup" );
  exit( 1 );
}
expect_deleteoldrdn = 1;
expect_newrdn = 0;
      } else {
fprintf( stderr, "%s: expecting \"%s:\" but saw \"%s:\" (line %d of entry %s)\n",
 prog, T_NEWRDNSTR, type, linenum, dn );
rc = LDAP_PARAM_ERROR;
      }
    } else if ( expect_deleteoldrdn ) {
      if ( strcasecmp( type, T_DELETEOLDRDNSTR ) == 0 ) {
deleteoldrdn = ( *value == '0' ) ? 0 : 1;
got_all = 1;
```

```
      } else {
fprintf( stderr, "%s: expecting \"%s:\" but saw \"%s:\" (line %d of entry %s)\n",
 prog, T_DELETEOLDRDNSTR, type, linenum, dn );
rc = LDAP_PARAM_ERROR;
      }
    } else if ( got_all ) {
      fprintf( stderr, "%s: extra lines at end (line %d of entry %s)\n",
       prog, linenum, dn );
      rc = LDAP_PARAM_ERROR;
    } else {
      addmodifyop(&pmods, modop, type, value, vlen, is_url, is_b64);
      type = NULL;
      value = NULL;
    }
  }

  /* If last separator is missing go ahead and handle it anyway, even
     though it is technically invalid ldif format. */
  if (expect_sep && (value != NULL))
    addmodifyop(&pmods, modop, value, NULL, 0, 0, 0);

  if ( rc == 0 ) {
    if (delete_entry)
      rc = dodelete( dn );
    else if (newrdn != NULL)
      rc = domodrdn( dn, newrdn, deleteoldrdn );
    else if (dn != NULL)
      rc = domodify( dn, pmods, new_entry );
  }

  if (dn != NULL)
    free( dn );
  if ( newrdn != NULL )
    free( newrdn );
  if ( pmods != NULL )
    ldap_mods_free( pmods, 1 );

  return rc;
}

/* Process a mod record. */
int process_ldapmod_rec( char *rbuf ) {
  char *line      = NULL;
  char *dn        = NULL;
  char *p         = NULL;
  char *q         = NULL;
  char *attr      = NULL;
  char *value     = NULL;
  int rc        = 0;
  int   linenum   = 0;
  int   modop     = 0;
  LDAPMod **pmods = NULL;

  while ( rc == 0 && rbuf != NULL && *rbuf != '\0' ) {
    ++linenum;
    if (( p = strchr( rbuf, '\n' )) == NULL ) {
      rbuf = NULL;
    } else {
      if ( *(p - 1) == '\\' ) { /* lines ending in '\' are continued */
strcpy( p - 1, p );
rbuf = p;
continue;
      }
      *p++ = '\0';
      rbuf = p;
    }
```

```
    if ( dn == NULL ) { /* first line contains DN */
      if (( dn = strdup( line )) == NULL ) {
perror( "strdup" );
exit( 1 );
      }
    } else {
      if (( p = strchr( line, '=' )) == NULL ) {
value = NULL;
p = line + strlen( line );
      } else {
*p++ = '\0';
value = p;
      }

      for ( attr = line; *attr != '\0' && isspace( *attr ); ++attr ) {
; /* skip attribute leading white space */
      }

      for ( q = p - 1; q > attr && isspace( *q ); --q ) {
*q = '\0'; /* remove attribute trailing white space */
      }

      if ( value != NULL ) {
while ( isspace( *value )) {
  ++value; /* skip value leading white space */
}
for ( q = value + strlen( value ) - 1; q > value &&
isspace( *q ); --q ) {
  *q = '\0'; /* remove value trailing white space */
}
if ( *value == '\0' ) {
  value = NULL;
}
      }

      if ((value == NULL) && (operation == LDAPMODIFY_ADD)) {
fprintf( stderr, "%s: missing value on line %d (attr is %s)\n",
 prog, linenum, attr );
rc = LDAP_PARAM_ERROR;
      } else {
switch ( *attr ) {
case '-':
  modop = LDAP_MOD_DELETE;
  ++attr;
  break;
case '+':
  modop = LDAP_MOD_ADD;
  ++attr;
  break;
default:
  modop = (operation == LDAPMODIFY_REPLACE)
    ? LDAP_MOD_REPLACE : LDAP_MOD_ADD;
  break;
}

addmodifyop( &pmods, modop, attr, value,
     ( value == NULL ) ? 0 : strlen( value ), 0, 0);
      }
    }
    line = rbuf;
  }

  if ( rc == 0 ) {
    if ( dn == NULL )
      rc = LDAP_PARAM_ERROR;
    else
      rc = domodify(dn, pmods, (operation == LDAPMODIFY_ADD));
```

```
  }

  if ( pmods != NULL )
    ldap_mods_free( pmods, 1 );
  if ( dn != NULL )
    free( dn );

  return rc;
}

main( int argc, char **argv ) {
  char *rbuf = NULL;
  char *start = NULL;
  char *p = NULL;
  char *q = NULL;
  char *tmpstr = NULL;
  int rc = 0;
  int   i = 0;
  int   use_ldif = 0;
  int   num_checked = 0;
  char  *Start_Transaction_OID      = LDAP_START_TRANSACTION_OID;
  char  *End_Transaction_OID        = LDAP_END_TRANSACTION_OID;
  char  *Control_Transaction_OID    = LDAP_TRANSACTION_CONTROL_OID;
  char  *Returned_OID               = NULL;
  struct berval *Returned_BerVal    = NULL;
  struct berval Request_BerVal      = {0,0};
  char  *Berval                     = NULL;
  LDAPMessage *LDAP_result          = NULL;

  /* Strip off any path info on program name */
#if defined( _WIN32 )
  if ((prog = strrchr(argv[0], '\\')) != NULL)
    ++prog;
  else
    prog = argv[0];
#else
  if (prog = strrchr(argv[0], '/'))
    ++prog;
  else
    prog = argv[0];
#endif

#if defined( _WIN32 )
  /* Convert string to lowercase */
  for (i = 0; prog[i] != '\0'; ++i)
    prog[i] = tolower(prog[i]);

  /* Strip ending .exe from program name */
  if ((tmpstr = strstr(prog, ".exe")) != NULL)
    *tmpstr = '\0';
#endif
  if ( strcmp( prog, "ldaptxadd" ) == 0 )
    operation = LDAPMODIFY_ADD;

  /* Parse command line arguments. */
  parse_arguments(argc, argv);

  /* Connect to server. */
  if (doit)
    connect_to_server();

  /* Disable translation if reading from file (they must specify the
     translation in the file). */
  if (fp != stdin)
    ldap_set_option(ld, LDAP_OPT_UTF8_IO, (void *)LDAP_UTF8_XLATE_OFF);

  /* Do the StartTransaction extended operation.
```

```
    The transaction ID returned must be put into the server control
    sent with all update operations. */
  rc = ldap_extended_operation_s ( ld, Start_Transaction_OID,
   &Request_BerVal, NULL, NULL,
   &Returned_OID,
   &Returned_BerVal);
  if (verbose) {
    printf("ldap_extended_operation(start transaction) RC=%d\n", rc);
  }

  if ( rc != LDAP_SUCCESS) {
    fprintf(stderr, "Start transaction rc=%d -> %s\n",
    rc, ldap_err2string(rc));
    exit( rc );
  }

  /* Allocate the server control for transactions. */
  if (( Server_Controls[0] =
(LDAPControl *)malloc( sizeof( LDAPControl ))) == NULL ) {
    perror("malloc");
    exit( 1 );
  }

  /* Allocate the server control's berval. */
  if ((Server_Controls[0]->ldctl_value.bv_val =
       (char *) calloc (1, Returned_BerVal->bv_len + 1)) == NULL) {
    perror("calloc");
    exit(1);
  }

  /* Copy the returned berval length and value into the server control */
  Server_Controls[0]->ldctl_value.bv_len = Returned_BerVal-> bv_len;
  memcpy(Server_Controls[0]->ldctl_value.bv_val,
 Returned_BerVal->bv_val , Returned_BerVal->bv_len);

  /* Set the control type to Transaction_Control_OID */
  Server_Controls[0]->ldctl_oid = Control_Transaction_OID;

  /* Set the criticality in the control to TRUE */
  Server_Controls[0]->ldctl_iscritical = LDAP_OPT_ON;

  /* If referral objects are to be modified directly, */
  if (manageDsa == LDAP_TRUE) {
    /* then set that server control as well. */
    Server_Controls[1] = &manageDsaIT
  }

  /* Initialize the count of operations that will be in the transaction.
     This count will be incremented by each operation that is performed.
     The count will be the number of calls that must be made to ldap_result
     to get the results for the operations.
  */
  Num_Operations = 0;
  /* Do operations */
  rc = 0;
  while ((rc == 0 || contoper) && (rbuf = read_one_record( fp )) != NULL ) {
    /* We assume record is ldif/slapd.replog if the first line
       has a colon that appears to the left of any equal signs, OR
       if the first line consists entirely of digits (an entry id). */

    use_ldif=1;
    start = rbuf;
    if ( use_ldif )
      rc = process_ldif_rec( start );
    else
      rc = process_ldapmod_rec( start );
    free( rbuf );
```

```
  }

  /* Finish the transaction, committing or rolling back based on input parameter. */
  rc = 0;
  Request_BerVal.bv_len = Returned_BerVal->bv_len + 1;
  if ((Berval =
       ( char *) malloc (Returned_BerVal->bv_len + 1)) == NULL) {
    perror("malloc");
    exit(1);
  }
  memcpy (&Berval[1], Returned_BerVal->bv_val, Returned_BerVal->bv_len);
  Berval[0] = abort_flag ? '\1' : '\0';
  Request_BerVal.bv_val = Berval;

  rc = ldap_extended_operation_s ( ld,
   End_Transaction_OID,
&Request_BerVal, NULL, NULL,
   &Returned_OID,
   &Returned_BerVal);
  if (verbose) {
    printf("ldap_extended_operation(end transaction) RC=%d\n", rc);
  }

  if ( rc != LDAP_SUCCESS) {
    fprintf(stderr, "End transaction rc=%d -> %s\n",
    rc, ldap_err2string(rc));
    exit( rc );
  }
  /* Process the results of the operations in the transaction.
     At this time we will not be concerned about the correctness
     of the message numbers, just whether the operations succceeded or not.
     We could keep track of the operation types and make sure they are all
     accounted for. */

  for ( num_checked = 0; num_checked < Num_Operations; num_checked++ ) {
    if (verbose) {
      printf("processing %d of %d operation results\n",
     1 + num_checked, Num_Operations);
    }

    rc = ldap_result (ld , LDAP_RES_ANY, LDAP_MSG_ONE, NULL, &LDAP_result);
    if ( rc <= 0) {
      if (rc == 0)
fprintf(stderr, "Operation %d timed out\n", num_checked);
      if (rc < 0 )
fprintf(stderr, "Operation %d failed\n", num_checked);
      exit( 1 );
    }
  }

  /* Unbind and exit */
  if (doit)
    ldap_unbind(ld);

  exit(0);
}
```

The following is an example makefile:

```
#-------------------------------------------------------------------------------
# COMPONENT_NAME: examples
#
# ABSTRACT: makefile to generate LDAP client programs for transactions
#
# ORIGINS: 202,27
#
# (C) COPYRIGHT International Business Machines Corp. 2002
```

```
# All Rights Reserved
# Licensed Materials - Property of IBM
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
#############################################################################
# Default definitions
#############################################################################
CC = cl.exe
LD      = link.exe
RM = erase /f
HARDLN = copy
### Note: Your install path may be different
LDAPHome = D:/Program Files/IBM/LDAP

#############################################################################
# General compiler options
#############################################################################

DEFINES = /DNDEBUG /DWIN32 /D_CONSOLE /D_MBCS /DNT /DNEEDPROTOS
INCLUDES= /I"$(LDAPHome)/include"
CFLAGS = /nologo /MD /GX /Z7 $(INCLUDES) $(DEFINES)

#############################################################################
# General linker options
#############################################################################

LIBS    = kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib\
 advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib\
 odbccp32.lib wsock32.lib

# Use the following definition to link the sample programs statically.
#CLIENT_LIBS = ldapstatic.lib libldif.lib setloci.lib iconvi.lib

# Use the following definition to link the sample programs with
# the LDAP shared library.
CLIENT_LIBS = ldap.lib libldif.lib setloci.lib
LDIR = /LIBPATH:"$(LDAPHome)"/lib
LFLAGS  = /nologo /subsystem:console /incremental:no  \
$(LDIR) $(LIBS) $(CLIENT_LIBS)

#############################################################################
# Targets
#############################################################################

all: ldaptxmod.exe ldaptxadd.exe

ldaptxmod.exe: ldaptxmod.obj
$(LD) $(LFLAGS) /out:$@ $**

ldaptxadd.exe: ldaptxmod.exe
$(RM) $@
$(HARDLN) ldaptxmod.exe ldaptxadd.exe

.c.obj::
   $(CC) $(CFLAGS) /c $<

ldaptxmod.obj: ldaptxmod.c

clean:
$(RM) ldaptxmod.exe ldaptxadd.exe ldaptxmod.obj
```

Chapter 7. LDAP client plug-in programming reference

Introduction to client SASL plug-ins

Client-side SASL plug-ins are used to extend the authentication capabilities of the LDAP client library. They work by intercepting the application's invocation of the ldap_sasl_bind_s() API. Note that SASL plug-ins are not designed to intercept asynchronous SASL binds.

Basic processing

The following describes the typical flow when a SASL plug-in is used to provide an extended authentication function. This flow assumes the SASL plug-in shared library has already been loaded by the LDAP library:

1. Application invokes ldap_sasl_bind_s(), with a mechanism supported by a configured SASL plug-in.
2. The LDAP library invokes the SASL bind worker function, as provided by the appropriate plug-in. The parameters supplied on the original ldap_sasl_bind_s() API are passed to the plug-in as elements of a pblock structure.
3. The plug-in's worker function receives control, and extracts the parameters from the pblock using the ldap_plugin_pblock_get() API. The following SASL-related information can be obtained from the pblock by the plug-in:
   - Distinguished Name (dn)
   - Credentials
   - Server controls
   - Client controls
   - Mechanism (plug-in subtype)

   In addition to these parameters, the plug-in can also obtain other information using the ldap_plugin_pblock_get(), including:
   - Plug-in configuration information (that is, configuration information supplied in ARGC and ARGV form)
   - Target LDAP server host name
4. The plug-in performs its mechanism-specific logic. Here are some sample mechanisms that can be implemented as SASL plug-ins, and thus be made available to all LDAP applications running on the system:

   Authentication based on a user's fingerprint (for example, mechanism=userfp)

   When the fingerprint plug-in gets control, it uses the DN supplied on the ldap_sasl_bind_s() API to obtain an image of the user's fingerprint. This can entail prompting the user to use a fingerprint scanning device. In this example, the fingerprint image, however obtained, represents the user's credentials.

   Once the credentials are obtained, the plug-in is ready to perform the actual SASL bind. This is done by invoking the ldap_plugin_sasl_bind_s() API, supplying the appropriate parameters (DN, credentials, mechanism, server controls). This is a synchronous API that sends the SASL bind request to the LDAP server. Two items are returned to the plug-in when the bind result is returned from the server, and control is returned to the plug-in:

- Bind result error code
- Server credentials

If the server credentials are to be returned to the application, they must be set in the pblock prior to returning control to the LDAP library, and subsequently to the application. This is done by using ldap_plugin_pblock_set(). In this example, the plug-in's work is complete, and it returns, supplying the bind result error code as the return code.

Authentication using credentials previously established by the operating system

When the plug-in gets control, it queries the local security context to obtain the user's identity and security token. For this example, we assume the user's identity, as associated with the local security context, is used to construct the DN, and information from the security token is used for credentials.

After the credentials are obtained, the plug-in invokes ldap_plugin_sasl_bind_s(), supplying the appropriate parameters (DN, credentials, mechanism, server controls). As in the previous example, the plug-in waits for the results of the bind request, then returns to the LDAP library, again setting server credentials in the pblock, if appropriate. Control is then returned to the application, along with the optional server credentials.

Authentication using multiple binds (mechanism=cram-md5)

Some SASL mechanisms require multiple transactions between the client and the server (for example, the SASL cram-md5 mechanism). For this type of mechanism, once the plug-in gains control, it actually invokes the ldap_plugin_sasl_bind_s() API multiple times. On each bind operation, the plug-in can supply DN, credentials, mechanism and server controls, which are passed to the server. The LDAP server can return a result and server credentials back to the client. The plug-in can use this information to formulate another bind, again sent to the server using ldap_plugin_sasl_bind_s(). Once the multi-bind flow is complete, the plug-in returns control to the LDAP library with the result and optional server credentials.

Restrictions

The plug-in must not use any LDAP APIs which accept ld as the input. This results in deadlock, since the ld is locked until the bind processing is complete.

Initializing a plug-in

A typical LDAP SASL plug-in contains two entry points:

- An initialization routine
- A worker routine, which implements the authentication function

When an instance of an application uses a SASL plug-in for the first time, the LDAP library obtains the configuration information for the plug-in. The configuration information can come from ldap.conf or might have been supplied explicitly by the application with the ldap_register_plugin() API.

Once the configuration information is located, the LDAP library loads the plug-in's shared library and invoke its initialization routine. By default, the name of the

initialization routine for a plug-in is ldap_plugin_init(). A different entry point can be defined in ldap.conf, or supplied on the ldap_plugin_register() API if the plug-in is explicitly registered by the application.

The plug-in's initialization routine is responsible for supplying the address of its worker routine's entry point, which actually implements the authentication function. This is done by using ldap_plugin_pblock_set() to define the address of the worker routine's entry point in the pblock. For example, the following code segment depicts a typical initialization routine, where authenticate_with_fingerprint is the name of the routine provided by the plug-in to perform a fingerprint-based authentication:

```
int ldap_plugin_init ( LDAP_Pblock      *pb )
{
        int rc;

        rc =  ldap_plugin_pblock_set ( pb, LDAP_PLUGIN_SASL_BIND_S_FN, ( void * )
              authenticate_with_fingerprint );
        if ( rc != LDAP_SUCCESS ) printf("ldap_plugin_init couldn't initialize
              worker function\n");
        return ( rc );
}
```

A pblock is an opaque structure in which parameters are stored. A pblock is used to communicate between the LDAP client library and a plug-in. The ldap_plugin_pblock_set and ldap_plugin_pblock_get APIs are provided for your plug-in to set, or get, parameters in the pblock structure.

Using ldap_plugin_pblock_get(), the plug-in can also access configuration parameters. For example, the following code segment depicts how the plug-in can access its configuration information:

```
int argc;
char ** argv;

rc = ldap_plugin_pblock_get ( pb, LDAP_PLUGIN_ARGC, &argc );
if (rc != LDAP_SUCCESS)
   return (rc);
rc = ldap_plugin_pblock_get( pb, LDAP_PLUGIN_ARGV, &argv );
if (rc != LDAP_SUCCESS)
   return (rc);
```

If the plug-in's initialization processing is significant, and the results need to be preserved and made available to the plug-in's worker function, the initialization routine can store the results of initialization as private instance data in its shared library. When the plug-in's worker function is subsequently invoked, it can access this private instance data. For example, during initialization, the plug-in might need to establish a session with a remote security server. Session information can be retained in the private instance data, which can be accessed later by the plug-in's worker function.

After your plug-in is correctly initialized, its worker function can be used by the LDAP library. Continuing the example shown above, if the mechanism supported by the plug-in is userfp, the authenticate_with_fingerprint function of your plug-in is invoked when the application issues an ldap_sasl_bind_s() function with mechanism="userfp". See "Sample worker function" on page 174 for an example of a plug-in's worker function.

Writing your own SASL plug-in

Do the following to write your own SASL plug-in:

1. Implement your own initialization and worker functions. Include ldap.h, where you can find all the parameters that can be obtained from the pblock, as well as the function prototypes for the available plug-in functions:
   - ldap_plugin_pblock_get()
   - ldap_plugin_pblock_set()
   - ldap_plugin_sasl_bind_s()
2. Identify the input parameters to your initialization and worker functions.

   Note: The LDAP library can pass parameters to your plug-in initialization function by way of the argument list that is specified in ldap.conf, or by way of the plugin_parmlist parameter on the ldap_register_plugin() API. Information might also be supplied as client-side controls.
3. The initialization function must call the ldap_plugin_pblock_set API in order to register your plug-in's worker function.
4. Implement your worker function. The worker function is responsible for obtaining the user's credentials and implementing the authentication function. Typically this involves invoking the ldap_plugin_sasl_bind_s() API one or more times. If the authentication is successful, LDAP_SUCCESS must be returned. Otherwise, the unsuccessful LDAP result must be returned as the return code. If appropriate, the worker function can also return a value for server credentials.
5. Export your initialization function from your plug-in library. Use an .exp file for the AIX operating system or Solaris operating system, or a .def (or dllexport) file for the Windows NT operating system to export your initialization function.
6. Compile your client plug-in functions. Set the include path to include ldap.h, and to link to ldap.lib. Compile and link all your LDAP plug-in object files with whatever libraries you need, including ldap.lib. Make sure that the initialization function is exported from the .dll you created.
7. Add a plug-in directive in the LDAP plug-in configuration file, ldap.conf. Alternatively, the application can define the plug-in by calling the ldap_register_plugin() API.

Plug-in APIs

For pblock access:

```
int ldap_plugin_pblock_get( LDAP_PBlock *pb, int arg, void **value );
int ldap_plugin_pblock_set( LDAP_PBlock *pb, int arg, void *value );
```

For sending an LDAP bind to the server:

```
int ldap_plugin_sasl_bind_s (
        LDAP            *ld,
        char            *dn,
        char            *mechanism,
        struct berval   *credentials,
        LDAPControl     **serverctrls,
        LDAPControl     **clientctrls,
        struct berval   **servercredp)
```

ldap_plugin_pblock_get()

The ldap_plugin_pblock_get() API returns the value associated with the specified pblock tag.

Syntax

```
#include "ldap.h"
int ldap_plugin_pblock_get( LDAP_PBlock *pb, int arg, void **value )
```

Parameters

pb Specifies the address of a pblock.

arg Specifies the tag or ID of the tag-value pair that you want to obtain from the pblock.

value Specifies a pointer to the address of the returned value.

Returns

Returns **0** if successful, or **-1** if an error occurs.

ldap_plugin_pblock_set()

The ldap_plugin_pblock_set API sets the value associated with the specified pblock tag.

Syntax

```
#include "ldap.h"
int ldap_plugin_pblock_set( LDAP_PBlock *pb, int arg, void *value );
```

Parameters

pb Specifies the address of a pblock.

arg Specifies the tag or ID of the tag-value pair that you want to set in the pblock.

value Specifies a pointer to the value that you want to set in the parameter block.

Returns

Returns **0** if successful, or **-1** if an error occurs.

ldap_plugin_sasl_bind_s()

The ldap_plugin_sasl_bind_s API is used by the plug-in to transmit an LDAP SASL bind operation to the LDAP server.

Syntax

```
#include "ldap.h"
int ldap_plugin_sasl_bind_s(
                        LDAP            *ld,
                        char            *dn,
                        char            *mechanism,
                        struct berval   *credentials,
                        LDAPControl     **serverctrls,
                        LDAPControl     **clientctrls,
                        struct berval   **servercredp)
```

Parameters

ld Specifies the LDAP pointer associated with the application's invocation of ldap_sasl_bind_s(). The plug-in obtains the LD with the ldap_plugin_pblock_get() API.

dn Specifies the Distinguished Name to bind the entry. The DN might have

been supplied by the application and obtained using ldap_plugin_pblock_get(), or it might have been obtained by other means.

credentials
Specifies the credentials to authenticate with. Arbitrary credentials can be passed using this parameter. The credentials might have been supplied by the application and obtained using ldap_plugin_pblock_get(), or they might have been obtained by other means.

mechanism
Specifies the SASL mechanism to be used when binding to the server. If a plug-in can be invoked for more than one mechanism, the plug-in can obtain the mechanism that was specified by the application with the ldap_plugin_pblock_get() API.

serverctrls
Specifies a list of LDAP server controls. See "LDAP controls" on page 45 for more information about server controls. The server controls might have been supplied by the application and obtained using ldap_plugin_pblock_get(), or they might have been obtained by other means.

clientctrls
Specifies a list of LDAP client controls. See "LDAP controls" on page 45 for more information about client controls.

Note: The client controls are not supported at this time for the ldap_plugin_sasl_bind_s() API.

Returns

error code
The error code is set to LDAP_SUCCESS if the bind succeeded. Otherwise it is set to a non-zero error code.

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

Sample worker function

```
/*  Sample SASL Plugin              */

#include <ldap.h>


int ldap_plugin_sasl_bind_s_prepare ( LDAP_Pblock   *pb )
{
        LDAP                    *ld;
        char                    *dn;
        char                    *mechanism;
        struct berval           *cred;
        LDAPControl             **serverctrls;
        LDAPControl             **clientctrls;
        struct berval           *servercredp = NULL;

        void *                  data;
        int                     rc;

        /*************************************************************************/
        /* Query pblock to obtain ld, dn, mechanism, credentials, server controls */
        /* and client controls, as supplied by application when it invoked the    */
        /* ldap_sasl_bind_s() API.                                                */
```

```
        /*****************************************************************************/

        if ( rc = ( ldap_plugin_pblock_get ( pb, LDAP_PLUGIN_LD, &data ))){
                printf( "Could not get parameter for bind operation\n" );
                return ( rc );
        }
        ld = ( LDAP * ) data;
        if ( rc = ( ldap_plugin_pblock_get ( pb, LDAP_PLUGIN_SASL_DN,
          &data )))
                return ( rc );
        dn = ( char * ) data;
        if ( rc = ( ldap_plugin_pblock_get ( pb, LDAP_PLUGIN_SASL_BIND_MECHANISM,
          &data )))
                return ( rc );
        mechanism = ( char * ) data;
        if ( rc = ( ldap_plugin_pblock_get ( pb, LDAP_PLUGIN_SASL_BIND_CREDENTIALS,
          &data )))
                return ( rc );
        cred = ( struct berval * ) data;
        if ( rc = ( ldap_plugin_pblock_get ( pb, LDAP_PLUGIN_SASL_BIND_SERVERCTRLS,
          &data )))
                return ( rc );
        serverctrls = ( LDAPControl ** ) data;
        if ( rc = ( ldap_plugin_pblock_get ( pb, LDAP_PLUGIN_SASL_BIND_CLIENTCTRLS,
          &data )))
                return ( rc );
        clientctrls = ( LDAPControl ** ) data;

        /*****************************************************************************/
        /* Perform plugin specific logic here to alter or obtain the user's       */
        /* distinguished name, credentials, etc.  This could include obtaining    */
        /* additional data from the pblock, including:                            */
        /*                                                                        */
        /*   LDAP_PLUGIN_TYPE       (e.g. "sasl")                                 */
        /*   LDAP_PLUGIN_ARGV       plugin config variables                       */
        /*   LDAP_PLUGIN_ARGC       plugin config variable count                  */
        /*                                                                        */
        /*****************************************************************************/

        if ( rc = ( ldap_plugin_sasl_bind_s (
                                             ld,
                                             dn,
                                             mechanism,
                                             cred,
                                             serverctrls,
                                             clientctrls,
                                             &servercredp)))
                return rc;

        data = ( void * ) servercredp;

        if ( rc = ( ldap_plugin_pblock_set ( pb, LDAP_PLUGIN_SASL_SERVER_CREDS,
          &data )))
                return rc;


        return ( LDAP_SUCCESS );
}


ldap_plugin_init ( LDAP_Pblock  *pb )
{
        int             argc;
        char            **argv;

        if ( rc = (ldap_plugin_pblock_set ( pb, LDAP_PLUGIN_SASL_BIND_S_FN,
                                        ( void * )
```

```
            ldap_plugin_sasl_bind_s_prepare )))
              return ( rc );

          return ( LDAP_SUCCESS );
}
```

Appendix A. LDAP V3 schema

Dynamic schema

The IBM Directory Server version 4.1 C-Client SDK requires that the schema defined for a server be stored in the directory's subschemasubentry.

To access the schema, you must first determine the subschemasubentry's DN, which is obtained by searching the root DSE. To obtain this information from the command-line, issue the following command:

```
ldapsearch -h hostname -p 389 -b "" -s base "objectclass=*"
```

The root DSE information returned from an LDAP V3 server, such as the IBM Directory server, includes the following:

```
subschemasubentry=cn=schema
```

where subschemasubentry's DN is ″cn=schema″.

Using the subschemasubentry's DN returned by searching the root DSE, schema information can be accessed with the following command-line search:

```
ldapsearch -h hostname -p 389 -b "cn=schema" -s base "objectclass=subschema"
```

The schema contains the following information:

Object class
A collection of attributes. A class can inherit attributes from one or more parent classes.

Attribute types
Contain information about the attribute, such as the name, oid, syntax and matching rules.

IBM attribute types
The IBM LDAP directory implementation-specific attributes, such as database table name, column name, SQL type, and the maximum length of each attribute.

Syntaxes
Specific LDAP syntaxes available for attribute definitions.

Matching rules
Specific matching rules available for attribute definitions.

Schema queries

The ldapsearch utility can be used to query the subschema entry. This search can be performed by any application using the ldap_search APIs.

To retrieve all the values of one or more selected attribute types, specify the specific attributes desired for the LDAP search. Schema-related attribute types include the following:

- objectclass
- objectclasses
- attributetypes

- ldapsyntaxes
- ibmattributetypes
- matchingrules

For example, to retrieve all the values for ldapsyntaxes, specify:

```
ldapsearch -h host -b "cn=schema" -s base objectclass=* ldapsyntaxes
```

which returns something like:

```
cn=schema
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.12 DESC 'DN' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.15 DESC 'Directory String' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.16 DESC 'DIT Content Rule
        Description' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.17 DESC 'DIT Structure Rule
        Description' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.24 DESC 'Generalized Time' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.26 DESC 'IA5 String' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.27 DESC 'INTEGER' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.3 DESC 'Attribute Type
        Description' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.30 DESC 'Matching Rule
        Description' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.31 DESC 'Matching Rule Use
        Description' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.35 DESC 'Name Form
        Description' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.37 DESC 'Object Class
        Description' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.38 DESC 'OID' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.5 DESC 'Binary' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.50 DESC 'Telephone
        Number' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.53 DESC 'UTC Time' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.54 DESC 'LDAP Syntax
        Description' )
ldapsyntaxes=( 1.3.6.1.4.1.1466.115.121.1.7 DESC 'Boolean' )
ldapsyntaxes=( IBMAttributeType-desc-syntax-oid DESC 'IBM Attribute
        Type Description' )
```

Similarly, to obtain the values for matchingrules, specify:

```
 ldapsearch -h host -b "cn=schema" -s base objectclass=* matchingrules
```

which returns something like:

```
cn=schema
MatchingRules= ( 2.5.13.5 NAME 'caseExactMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
     MatchingRules= ( 2.5.13.2 NAME 'caseIgnoreMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
     MatchingRules= ( 2.5.13.7 NAME 'caseExactSubstringsMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
     MatchingRules= ( 2.5.13.6 NAME 'caseExactOrderingMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
     MatchingRules= ( 2.5.13.4 NAME 'caseIgnoreSubstringsMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
     MatchingRules= ( 2.5.13.3 NAME 'caseIgnoreOrderingMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
     MatchingRules= ( 1.3.18.0.2.4.405 NAME 'distinguishedNameOrderingMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 )
     MatchingRules= ( 2.5.13.1 NAME 'distinguishedNameMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.12 )
     MatchingRules= ( 2.5.13.28 NAME 'generalizedTimeOrderingMatch' \
       SYNTAX 1.3.6.1.4.1.1466.115.121.1.24 )
     MatchingRules= ( 2.5.13.27 NAME 'generalizedTimeMatch' \
```

```
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.24 )
MatchingRules= ( 1.3.6.1.4.1.1466.109.114.2 NAME 'caseIgnoreIA5Match' \
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
MatchingRules= ( 1.3.6.1.4.1.1466.109.114.1 NAME 'caseExactIA5Match' \
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
MatchingRules= ( 2.5.13.29 NAME 'integerFirstComponentMatch' \
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 )
MatchingRules= ( 2.5.13.14 NAME 'integerMatch' \
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.27 )
MatchingRules= ( 2.5.13.17 NAME 'octetStringMatch' \
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.5 )
MatchingRules= ( 2.5.13.0 NAME 'objectIdentifierMatch' \
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.38 )
MatchingRules= ( 2.5.13.30 NAME 'objectIdentifierFirstComponentMatch' \
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.38 )
MatchingRules= ( 2.5.13.21 NAME 'telephoneNumberSubstringsMatch' \
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.50 )
MatchingRules= ( 2.5.13.20 NAME 'telephoneNumberMatch' \
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.50 )
MatchingRules= ( 2.5.13.25 NAME 'uTCTimeMatch' \
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.53 )
```

Dynamic schema changes

To perform a dynamic schema change, use LDAP modify with a DN of "cn=schema". It is permissible to add, delete or replace only one schema entity, for example, an attribute type or an object class, at a time.

To delete a schema entity, you can simply provide the oid in parentheses:
`( oid )`

A full description might also be provided. In either case, the matching rule used to find the schema entity to delete is objectIdentifierFirstComponentMatch as mandated by the LDAP V3 protocol.

To add or replace a schema entity, you must provide the LDAP V3 definition and you can provide the IBM definition.

In all cases, you must only provide the definitions of the schema entity you wish to affect. For example, to delete the attribute type cn (its OID is 2.5.4.3), invoke ldap_modify() with:

```
LDAPMod  attr;
LDAPMod *attrs[] = { &attr, NULL };
char    *vals [] = { "( 2.5.4.3 )", NULL };
attr.mod_op      = LDAP_MOD_DELETE;
attr.mod_type    = "attributeTypes";
attr.mod_values  = vals;
ldap_modify_s(ldap_session_handle, "cn=schema", attrs);
```

To add a new attribute type foo with OID 20.20.20 which is a NAME of length 20 chars:

```
char    *vals1[] = { "( 20.20.20 NAME 'foo' SUP NAME )", NULL };
char    *vals2[] = { "( 20.20.20 LENGTH 20 )", NULL };
LDAPMod  attr1;
LDAPMod  attr2;
LDAPMod *attrs[] = { &attr1, &attr2, NULL };
attr1.mod_op = LDAP_MOD_ADD;
attr1.mod_type = "attributeTypes";
attr1.mod_values = vals1;
attr2.mod_op = LDAP_MOD_ADD;
```

```
attr2.mod_type = "IBMattributeTypes";
attr2.mod_values = vals2;
ldap_modify_s(ldap_session_handle, "cn=schema", attrs);
```

To change the object class top so it allows a MAY attribute type called foo (this assumes the attribute type foo has been defined in the schema):

```
LDAPMod  attr;
LDAPMod *attrs[] = { &attr, NULL };
attr.mod_op = LDAP_MOD_REPLACE;
attr.mod_type = "objectClasses";
attr.mod_values = "( 2.5.6.0 NAME 'top' ABSTRACT "
                  "MUST objectClass MAY foo )";
ldap_modify_s(ldap_session_handle, "cn=schema", attrs);
```

Appendix B. LDAP distinguished names

Distinguished names (DNs) are used to uniquely identify entries in an LDAP or X.500 directory. DNs are user-oriented strings, typically used whenever you must add, modify or delete an entry in a directory using the LDAP programming interface, as well as when using the LDAP utilities ldapmodify, ldapsearch, ldapmodrdn and ldapdelete.

A DN is typically composed of an ordered set of attribute type/attribute value pairs. Most DNs are composed of pairs in the following order:

- common name (cn)
- organization (o) or organizational unit (ou)
- country (c)

The following string-type attributes represent the set of standardized attribute types for accessing an LDAP directory. A DN can be composed of attributes with an LDAP syntax of Directory String, including the following:

- CN - CommonName
- L - LocalityName
- ST - StateOrProvinceName
- O - OrganizationName
- OU - OrganizationalUnitName
- C - CountryName
- STREET - StreetAddress

Informal definition

This notation is designed to be convenient for common forms of name. Most DNs begin with CommonName (CN), and progress up the naming tree of the directory. Typically, as you read from left to right, each component of the name represents increasingly larger groupings of entries, ending with CountryName (C). Remember that sequence is important. For example, the following two DNs do not identify the same entry in the directory:

```
CN=wiley coyote, O=acme, O=anvils, C=US

CN=wiley coyote, O=anvils, O=acme, C=US
```

Some examples follow. The author of RFC 2253, "UTF-8 String Representation of Distinguished Names" is specified as:

```
CN=Steve Kille, O=ISODE Consortium, C=GB
```

Another name might be:

```
CN=Christian Huitema, O=INRIA, C=FR
```

A semicolon (;) can be used as an alternate separator. The separators might be mixed, but this usage is discouraged.

```
CN=Christian Huitema; O=INRIA; C=FR
```

Here is an example of a multi-valued Relative Distinguished Name, where the namespace is flat within an organization, and department is used to disambiguate certain names:

```
OU=Sales + CN=J. Smith, O=Widget Inc., C=US
```

The final examples show both methods of entering a comma in an Organization name:

```
CN=L. Eagle, O="Sue, Grabbit and Runn", C=GB
```

```
CN=L. Eagle, O=Sue, Grabbit and Runn, C=GB
```

Formal definition

For a formal, and more complete, definition of Distinguished Names that can be used with the LDAP interfaces, see ″RFC 2253, UTF-8 String Representation of Distinguished Names″.

Appendix C. LDAP data interchange format (LDIF)

This documentation describes the LDAP Data Interchange Format (LDIF), as used by the ldapmodify, ldapsearch and ldapadd utilities. The LDIF specified here is also supported by the server utilities provided with the IBM Directory.

LDIF is used to represent LDAP entries in text form. The basic form of an LDIF entry is:

```
dn: <distinguished name>
<attrtype> : <attrvalue>
<attrtype> : <attrvalue>
...
```

A line can be continued by starting the next line with a single space or tab character, for example:

```
dn: cn=John E Doe, o=University of High
 er Learning, c=US
```

Multiple attribute values are specified on separate lines, for example:

```
cn: John E Doe
cn: John Doe
```

If an <*attrvalue*> contains a non-US-ASCII character, or begins with a space or a colon (:), the <*attrtype*> is followed by a double colon and the value is encoded in base-64 notation. For example, the value `begins with a space` is encoded as:

```
cn:: IGJlZ2lucyB3aXRoIGEgc3BhY2U=
```

Multiple entries within the same LDIF file are separated by a blank line. Multiple blank lines are considered a logical end-of-file.

LDIF example

Here is an example of an LDIF file containing three entries.

```
dn: cn=John E Doe, o=University of High
 er Learning, c=US
cn: John E Doe
cn: John Doe
objectclass: person
sn: Doe

dn: cn=Bjorn L Doe, o=University of High
 er Learning, c=US
cn: Bjorn L Doe
cn: Bjorn Doe
objectclass: person
sn: Doe

dn: cn=Jennifer K. Doe, o=University of High
 er Learning, c=US
cn: Jennifer K. Doe
cn: Jennifer Doe
objectclass: person
sn: Doe
jpegPhoto:: /9j/4AAQSkZJRgABAAAAAQABAAD/2wBDABALD
 A4MChAODQ4SERATGCgaGBYWGDEjJR0oOjM9PDkzODdASFxOQ
 ERXRTc4UG1RV19iZ2hnPk1xeXBkeFxlZ2P/2wBDARESEhgVG
...
```

The jpegPhoto in Jennifer Doe's entry is encoded using base-64. The textual attribute values can also be specified in base-64 format. However, if this is the case, the base-64 encoding must be in the code page of the wire format for the protocol, that is, for LDAP V2, the IA5 character set and for LDAP V3, the UTF-8 encoding.

Version 1 LDIF support

The client utilities (ldapmodify and ldapadd) have been enhanced to recognize the latest version of LDIF, which is identified by the presence of the version: 1 tag at the head of the file. Unlike the original version of LDIF, the newer version of LDIF supports attribute values represented in UTF-8, instead of the very limited US-ASCII.

However, manual creation of an LDIF file containing UTF-8 values can be difficult. In order to simplify this process, a charset extension to the LDIF format is supported. This extension allows an IANA character set name to be specified in the header of the LDIF file, along with the version number. A limited set of the IANA character sets are supported. See "IANA character sets supported by platform" on page 185 for the specific charset values that are supported for each operating system platform.

The version 1 LDIF format also supports file URLs. This provides a more flexible way to define a file specification. File URLs take the following form:

```
attribute:< file:///path
    (where path syntax depends on platform)
```

For example, the following are valid file Web addresses:

```
jpegphoto:< file:///d:\temp\photos\myphoto.jpg
     (DOS/Windows style paths)
jpegphoto:< file:///etc/temp/photos/myphoto.jpg
     (Unix style paths)
```

Note: The IBM Directory Server utilities support both the new file URL specification as well as the older style, for example, jpegphoto: /etc/temp/myphoto, regardless of the version specification. In other words, the new file URL format can be used without adding the version tag to your LDIF files.

Version 1 LDIF examples

You can use the optional charset tag so that the utilities automatically convert from the specified character set to UTF-8 as in the following example:

```
version: 1
charset: ISO-8859-1

dn: cn=Juan Griego, o=University of New Mexico, c=US
cn: Juan Griego
sn: Griego
description:: V2hhdCBhIGNhcmVmdWwgcmVhZGVyIHlvd
title: Associate Dean
title: [title in Spanish]
jpegPhoto:> file:///usr/local/photos/jgriego.jpg
```

In this instance, all values following an attribute name and a single colon are translated from the ISO-8859-1 character set to UTF-8. Values following an attribute name and a double colon (such as description:: V2hhdCBhIGNhcm...) must be base-64 encoded, and are expected to be either binary or UTF-8 character strings.

Values read from a file, such as the jpegPhoto attribute specified by the Web address in the previous example, are also expected to be either binary or UTF-8. No translation from the specified charset to UTF-8 is done on those values.

In this example of an LDIF file without the charset tag, content is expected to be in UTF-8, or base-64 encoded UTF-8, or base-64 encoded binary data:

```
# IBM Directory sample LDIF file
 #
 # The suffix "o=IBM, c=US" should be defined before attempting to load
 # this data.

 version: 1

 dn: o=IBM, c=US
 objectclass: top
 objectclass: organization
 o: IBM

 dn: ou=Austin, o=IBM, c=US
 ou: Austin
 objectclass: organizationalUnit
 seealso: cn=Linda Carlesberg, ou=Austin, o=IBM, c=US
```

This same file can be used without the version: 1 header information, as in previous releases of the IBM Directory Server version 4.1 C-Client SDK:

```
 # IBM Directory sample LDIF file
 #
 # The suffix "o=IBM, c=US" should be defined before attempting to load
 # this data.

 dn: o=IBM, c=US
 objectclass: top
 objectclass: organization
 o: IBM

 dn: ou=Austin, o=IBM, c=US
 ou: Austin
 objectclass: organizationalUnit
 seealso: cn=Linda Carlesberg, ou=Austin, o=IBM, c=US
```

Note: The textual attribute values can be specified in base-64 format.

IANA character sets supported by platform

The following table defines the set of Internet Assigned Numbers Authority (IANA)-defined character sets that can be defined for the charset tag in a Version 1 LDIF file, on a per-platform basis. The value in the left-most column defines the text string that can be assigned to the charset tag. An **X** indicates that conversion from the specified charset to UTF-8 is supported for the associated platform, and that all string content in the LDIF file is assumed to be represented in the specified charset. **n/a** indicates that the conversion is not supported for the associated platform.

String content is defined to be all attribute values that follow an attribute name and a single colon.

See IANA Character Sets for more information about IANA-registered character sets.

Table 1.

Character	Conversion Supported			
Set Name	**NT**	**AIX**	**Solaris**	**Linux**
ISO-8859–1	X	X	X	X
ISO-8859–2	X	X	X	X
ISO-8859–5	X	X	X	X
ISO-8859–6	X	X	X	X
ISO-8859–7	X	X	X	X
ISO-8859–8	X	X	X	X
ISO-8859–9	X	X	X	X
ISO-8859–15	NA	X	X	
IBM437	X	NA	NA	
IBM850	X	X	NA	
IBM852	X	NA	NA	
IBM857	X	NA	NA	
IBM862	X	NA	NA	
IBM864	X	NA	NA	
IBM866	X	NA	NA	
IBM869	X	X	NA	
IBM1250	X	NA	NA	
IBM1251	X	NA	NA	
IBM1253	X	NA	NA	
IBM1254	X	NA	NA	
IBM1255	X	NA	NA	
IBM1256	X	NA	NA	
TIS-620	X	X	NA	
EUC-JP	NA	X	X	X
EUC-KR	NA	X	X*	
EUC-CN	NA	X	X	
EUC-TW	NA	X	X	
Shift-JIS	X	X	X	X
KSC	X	X	NA	
GBK	X	X	X*	
Big5	X	X	X	

* Supported on Solaris 7 and higher only.

The new Chinese character set standard (GB18030) is supported on the following platforms with appropriate patches available from www.sun.com and www.microsoft.com:

- Windows 2000
- AIX
- Solaris

Note: On Windows 2000, you must set the environment variable zhCNGB18030=TRUE.

Appendix D. Deprecated LDAP APIs

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

- ldap_ssl_start()—use ldap_ssl_client_init() and ldap_ssl_init(). See “LDAP_SSL” on page 107.
- ldap_open()—use ldap_init(). See“LDAP_INIT” on page 59.
- ldap_bind()—use ldap_simple_bind(). See “LDAP_BIND / UNBIND” on page 36.
- ldap_bind_s()—use ldap_simple_bind_s(). See “LDAP_BIND / UNBIND” on page 36.
- ldap_modrdn()—use ldap_rename(). See “LDAP_RENAME” on page 82.
- ldap_modrdn_s()—use ldap_rename_s(). See “LDAP_RENAME” on page 82.
- ldap_result2error()—use ldap_parse_result(). See “LDAP_PARSE_RESULT” on page 76.
- ldap_perror()—use ldap_parse_result(). See “LDAP_PARSE_RESULT” on page 76.
- ldap_get_entry_controls_np—use ldap_get_entry_controls. See “LDAP_FIRST_ENTRY/REFERENCE” on page 47.
- ldap_parse_reference_np—use ldap_parse_reference. See “LDAP_FIRST_ENTRY/REFERENCE” on page 47.

Appendix E. Notices

This information was developed for products and services offered in the U.S.A. IBM might 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 in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Department LZKS
11400 Burnet Road
Austin, TX 78758
U.S.A.

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

The licensed program described in this document 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.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both: AIX, IBM, SecureWay, Tivoli, World Registry.

Microsoft, Windows, and Windows NT are registered trademarks of Microsoft Corporation.

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

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

Index

M

N

O

P

R

S

T

U

V

Printed in U.S.A.