

IBM Tivoli Directory Server



C-Client SDK Programming Reference

Version 6.0

IBM Tivoli Directory Server



C-Client SDK Programming Reference

Version 6.0

Note

Before using this information and the product it supports, read the general information under Appendix L, "Notices," on page 301.

First Edition (April 2005)

This edition applies to version 6, release 0, of the IBM Tivoli Directory Server and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright International Business Machines Corporation 2005. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Preface ix

Who should read this book	ix
Publications	ix
IBM Tivoli Directory Server library.	ix
Related publications	ix
Accessing publications online.	x
Ordering publications	x
Accessibility	x
Tivoli technical training	xi
Support information	xi
Conventions used in this book	xi
Typeface conventions	xi
Operating system differences.	xi

Chapter 1. IBM Directory C-Client SDK

overview 1

LDAP version support	1
LDAP API overview	1
Typical API usage	2
Displaying results	3
Uniform Resource Locators (URLs)	3
Secure Socket Layer (SSL) support	3
Updates for IBM Tivoli Directory Server C-Client	
Version 6.0	3
New client utilities	3
New APIs	3
Kerberos.	3

Chapter 2. LDAP utilities 5

idslsapmodify, ldapmodify, idslsapadd, ldapadd	5
Synopsis	5
Description	5
Options	6
Input format	9
Alternative input format	9
Examples	9
Notes	11
Security functions	11
Diagnostics	11
See also.	11
idslsapchangepwd, ldapchangepwd	11
Synopsis	11
Description	11
Options.	11
Examples	14
Security functions	14
Diagnostics	14
See also.	14
idslsapdelete, ldapdelete	14
Synopsis	14
Description	14
Options.	14
Examples	17
Notes	17
Security functions	17

Diagnostics	17
See also.	17
idslsapdiff, ldapdiff	18
Synopsis	18
Description	18
Options.	21
Examples	26
Notes	27
Security functions	27
Diagnostics	27
idslsapexop, ldapexop.	27
Synopsis	27
Description	27
Options.	27
Notes	35
Security functions	35
Diagnostics	35
User type and user roles for extended operations	35
See also.	37
idslsapmodrdn, ldapmodrdn	37
Synopsis	37
Description	37
Options.	37
Input format for dn newrdn.	40
Examples	40
Notes	41
Security functions	41
Diagnostics	41
See also.	41
idslsapsearch, ldapsearch.	41
Synopsis	41
Description	41
Options.	42
Output format	47
Examples	47
Security functions	50
Diagnostics	50
See also.	50
idslsaptrace, ldaptrace.	50
Synopsis	50
Description	50
Options.	51
Examples	53
See also.	54
ibmdirctl, idsdirctl	54
Synopsis	54
Description	54
Options.	54
Example	55
SSL, TLS notes	55

Chapter 3. API categories. 57

LDAP_ABANDON	57
Purpose	57
Synopsis	58

Input parameters	58	Purpose	88
Usage	58	Synopsis	88
Errors	58	Input parameters	89
See also.	59	Output parameters	89
LDAP_ADD	59	Usage	89
Purpose	59	Errors	90
Synopsis	59	Notes	90
Input parameters	59	See also.	90
Output parameters	60	LDAP_FIRST_ATTRIBUTE	90
Usage	60	Purpose	90
Errors	60	Synopsis	90
See also.	60	Input parameters	91
LDAP_BIND / UNBIND	61	Output parameters	91
Purpose	61	Usage	91
Synopsis	61	Errors	92
Input parameters	62	Notes	92
Output parameters	63	See also.	92
Usage	64	LDAP_FIRST_ENTRY, LDAP_FIRST_REFERENCE	92
Errors	68	Purpose	92
See also.	68	Synopsis	92
LDAP_CODEPAGE	68	Input parameters	93
Purpose	69	Usage	93
Synopsis	69	Errors	94
Input parameters	69	See also.	95
Output parameters	70	LDAP_GET_BIND_CONTROLS	95
Usage	71	Purpose	95
Errors	74	Synopsis	95
See also.	74	Input parameters	95
LDAP_CREATE_PROXYAUTH_CONTROL	74	Output parameters	95
Purpose	74	Usage	95
Synopsis	75	Errors	95
Input parameters	75	See also.	95
Usage	75	LDAP_GET_DN	95
Errors	76	Purpose	96
See also.	76	Synopsis	96
LDAP_COMPARE	76	Input parameters	96
Purpose	76	Usage	96
Synopsis	76	Errors	98
Input parameters	77	Notes	98
Output parameters	77	See also.	98
Usage	77	LDAP_GET_VALUES	98
Errors	78	Purpose	98
See also.	78	Synopsis	98
LDAP controls	78	Input parameters	99
Functions to manipulate controls	78	Usage	99
LDAP_DELETE	80	Errors	100
Purpose	80	See also	100
Synopsis	80	LDAP_INIT	100
Input parameters	80	Purpose	100
Output parameters	81	Synopsis	100
Usage	81	Input parameters	100
Errors	81	Usage	103
See also.	81	Errors	109
LDAP_ERROR	81	LDAP_DEBUG	109
Purpose	82	LDAP_SET_OPTION syntax for LDAP V2	
Synopsis	82	applications	109
Input parameters	82	Locating default LDAP servers	110
Usage	83	Multithreaded applications	111
Errors	83	Notes	111
See also.	88	See also	111
LDAP_EXTENDED_OPERATION	88	LDAP_MEMFREE	111

Purpose	112	Synopsis	130
Synopsis	112	Input parameters	130
Input parameters	112	Output parameters	130
Usage	112	Usage	131
See also	113	Errors	131
LDAP_MESSAGE	113	Notes	131
Purpose	113	See also	131
Synopsis	113	LDAP_SEARCH	131
Input parameters	113	Purpose	131
Usage	113	Synopsis	131
Errors	114	Input parameters	132
See also	114	Output parameters	133
LDAP_MODIFY	114	Usage	134
Purpose	114	Errors	135
Synopsis	114	Notes	135
Input parameters	115	See also	135
Output parameters	115	LDAP_SERVER_INFORMATION IN DNS	135
Usage	115	Purpose	135
Errors	117	Synopsis	136
See also	117	Input parameters	136
LDAP_PAGED_RESULTS	117	Output parameters	140
Purpose	117	Usage	142
Synopsis	117	Errors	152
Input parameters	118	See also	152
Output parameters	118	LDAP_SSL	152
Usage	118	Purpose	152
Errors	120	Synopsis	152
Notes	120	Input parameters	153
See also	120	Usage	157
LDAP_PARSE_RESULT	120	Options	159
Purpose	120	Notes	159
Synopsis	121	See also	159
Input parameters	121	LDAP_START_TLS	159
Usage	122	Purpose	159
Errors	123	Synopsis	160
See also	123	Input parameters	160
LDAP_PASSWORD_POLICY	123	Usage	160
Purpose	123	Errors	160
Synopsis	123	See also	160
Input parameters	123	LDAP_STOP_TLS	161
Usage	123	Purpose	161
Errors	124	Synopsis	161
See also	124	Input parameters	161
LDAP_PLUGIN_REGISTRATION	124	Usage	161
Purpose	124	Errors	161
Synopsis	124	See also	161
Input parameters	125	LDAP_URL	161
Output parameters	125	Purpose	161
Usage	125	Synopsis	161
Errors	127	Input parameters	162
See also	127	Output parameters	162
LDAP_RENAME	127	Usage	162
Purpose	127	Notes	163
Synopsis	127	See also	164
Input parameters	128	LDAP_SSL_ENVIRONMENT_INIT	164
Output parameters	128	Purpose	164
Usage	129	Synopsis	164
Errors	129	LDAP_SORT	165
See also	129	Purpose	165
LDAP_RESULT	129	Synopsis	165
Purpose	130	Input parameters	165

Output parameters	166
Usage	166
Errors	169
Notes	169
See also	170

Chapter 4. Using gsk7IKM 171

Creating a key pair and requesting a certificate from a Certificate Authority	171
Receiving a certificate into a key database	173
Changing a key database password	173
Showing information about a key	174
Deleting a key	174
Making a key the default key in the key database	175
Creating a key pair and certificate request for self-signing	175
Exporting a key	176
Importing a key	177
Designating a key as a trusted root	177
Removing a key as a trusted root.	178
Requesting a certificate for an existing key	178
Migrating a keyring file to the key database format	179

Chapter 5. Event notification 181

Registration request	181
Registration response.	181
Usage	182
Unregistering a client.	182
Example	182

Chapter 6. LDAP client plug-in programming reference 185

Introduction to client SASL plug-ins.	185
Basic processing	185
Restrictions	186
Initializing a plug-in	186
Writing your own SASL plug-in	188
Plug-in APIs.	188
ldap_plugin_pblock_get()	188
ldap_plugin_pblock_set()	189
ldap_plugin_sasl_bind_s()	189
Sample worker function	190

Appendix A. Possible extended error codes returned by LDAP SSL function codes. 193

Appendix B. LDAP V3 schema 197

Dynamic schema	197
Schema queries.	197
Dynamic schema changes	199

Appendix C. LDAP distinguished names 201

Informal definition	201
Formal definition	202

Appendix D. LDAP data interchange format (LDIF) 203

LDIF examples	203
LDIF example: Content	203
LDIF file: Change types	204
Version 1 LDIF support	205
Version 1 LDIF examples	205
IANA character sets supported by platform	206

Appendix E. Deprecated LDAP APIs 209

Appendix F. Object Identifiers (OIDs) for extended operations and controls . 211

OIDs for extended operations	211
Account status extended operation	213
Attribute type extended operations	214
Begin transaction extended operation	216
Cascading replication operation extended operation.	217
Control replication extended operation	218
Control queue extended operation	220
DN normalization extended operation	221
Dynamic server trace extended operation	222
Dynamic update requests extended operation	223
End transaction extended operation	224
Event notification register request extended operation.	225
Event notification unregister request extended operation.	226
Group evaluation extended operation	227
Kill connection extended operation	228
LDAP trace facility extended operation.	229
Quiesce or unquiesce replication context extended operation	230
Replication error log extended operation	232
Replication topology extended operation	233
Start, stop server extended operations	234
Start TLS extended operation	235
Unique attributes extended operation	236
Update configuration extended operation	237
Update event notification extended operation	239
Update log access extended operation	239
User type extended operation	240
Log access extended operations	241
OIDs for controls	244
AES bind control	246
Audit control	247
Do not replicate control	248
Group authorization control	248
Manage DSAIT control	250
Modify groups only control	250
No replication conflict resolution control	251
Omit group referential integrity control.	252
Paged search results control	252
Password policy request control	254
Proxy authorization control.	255
Refresh entry control	256
Replication supplier bind control	257
Replication update ID control	258
Server administration control	258

Sorted search results control	259	Searching knowledge bases.	297
Subtree delete control	260	Search the information center on your local system or network.	297
Transaction control	261	Search the Internet	297
Appendix G. Building LDAP-enabled applications	263	Obtaining fixes	297
Appendix H. Client libraries	265	Contacting IBM Software Support	298
Appendix I. Sample Makefile	267	Determine the business impact of your problem	299
Appendix J. Limited transaction support	271	Describe your problem and gather background information	299
Usage	271	Submit your problem to IBM Software Support	299
Example	272	Appendix L. Notices	301
Appendix K. Support information	297	Trademarks	303
		Index	305

Preface

This book contains information about writing LDAP client applications, including:

- Various sample LDAP client programs
- An LDAP client library that is used to provide application access to the LDAP servers

Who should read this book

This book is intended for programmers.

Publications

Read the descriptions of the IBM® Tivoli® Directory Server library, the prerequisite publications, and the related publications to determine which publications you might find helpful. After you determine the publications you need, see “Accessing publications online” on page x for information about accessing publications online.

IBM Tivoli Directory Server library

The publications in the IBM Tivoli Directory Server library are:

IBM Tivoli Directory Server Version 6.0 Release Notes

Contains information about the new features in the IBM Tivoli Directory Server Version 6.0 release.

IBM Tivoli Directory Server Version 6.0 Installation and Configuration Guide

Contains complete information for installing the IBM Tivoli Directory Server client, server, and Web Administration Tool. Includes information about migrating from a previous version of IBM Tivoli Directory Server or SecureWay® Directory.

IBM Tivoli Directory Server Version 6.0 Performance Tuning Guide

Contains information about tuning your server for better performance.

IBM Tivoli Directory Server Version 6.0 Administration Guide

Contains instructions for performing administrator tasks through the Web Administration Tool and the command line.

IBM Tivoli Directory Server Version 6.0 Plug-ins Reference

Contains information about writing server plug-ins.

IBM Tivoli Directory Server Version 6.0 C-Client SDK Programming Reference

Contains information about writing Lightweight Directory Access Protocol (LDAP) client applications.

IBM Tivoli Directory Server Version 6.0 Problem Determination Guide

Contains information about possible problems and corrective actions that can be tried before contacting Software Support.

IBM Tivoli Directory Server Version 6.0 Messages

Contains information about error messages that you might see.

Related publications

Information related to IBM Tivoli Directory Server is available in the following publications:

- IBM Tivoli Directory Server Version 6.0 uses the JNDI client from Sun Microsystems. For information about the JNDI client, refer to the *Java™ Naming and Directory Interface™ 1.2.1 Specification* on the Sun Microsystems Web site at <http://java.sun.com/products/jndi/1.2/javadoc/index.html>.
- The Tivoli Software Library provides a variety of Tivoli publications such as white papers, datasheets, demonstrations, redbooks, and announcement letters. The Tivoli Software Library is available on the Web at: <http://www.ibm.com/software/tivoli/library/>
- The *Tivoli Software Glossary* includes definitions for many of the technical terms related to Tivoli software. The *Tivoli Software Glossary* is available from the **Glossary** link on the left side of the Tivoli Software Library Web page <http://www.ibm.com/software/tivoli/library/>

Accessing publications online

The publications for this product are available online in Portable Document Format (PDF) or Hypertext Markup Language (HTML) format, or both in the Tivoli software library: <http://www.ibm.com/software/tivoli/library>

To locate product publications in the library, click the **Product manuals** link on the left side of the library page. Then, locate and click the name of the product on the Tivoli software information center page.

Product publications include release notes, installation guides, user's guides, administrator's guides, and developer's references.

Note: To ensure proper printing of PDF publications, select the **Fit to page** check box in the Adobe Acrobat Print window (which is available when you click **File** → **Print**).

Ordering publications

You can order many Tivoli publications online at the following Web site:

<http://www.elink.ibm.com/public/applications/publications/cgi-bin/pbi.cgi>

You can also order by telephone by calling one of these numbers:

- In the United States: 800-879-2755
- In Canada: 800-426-4968

In other countries, see the following Web site for a list of telephone numbers:

<http://www.ibm.com/software/tivoli/order-lit/>

Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. With this product, you can use assistive technologies to hear and navigate the interface. You also can use the keyboard instead of the mouse to operate all features of the graphical user interface.

Tivoli technical training

For Tivoli technical training information, refer to the IBM Tivoli Education Web site: <http://www.ibm.com/software/tivoli/education>.

Support information

If you have a problem with your IBM software, you want to resolve it quickly. IBM provides the following ways for you to obtain the support you need:

- Searching knowledge bases: You can search across a large collection of known problems and workarounds, Technotes, and other information.
- Obtaining fixes: You can locate the latest fixes that are already available for your product.
- Contacting IBM Software Support: If you still cannot solve your problem, and you need to work with someone from IBM, you can use a variety of ways to contact IBM Software Support.

For more information about these three ways of resolving problems, see Appendix K, “Support information,” on page 297.

Conventions used in this book

This reference uses several conventions for special terms and actions and for operating system-dependent commands and paths.

Typeface conventions

The following typeface conventions are used in this reference:

Bold Lowercase commands or mixed case commands that are difficult to distinguish from surrounding text, keywords, parameters, options, names of Java classes, and objects are in **bold**.

Italic Variables, titles of publications, and special words or phrases that are emphasized are in *italic*.

<*Italic*>

Variables are set off with < > and are in <*italic*>.

Monospace

Code examples, command lines, screen output, file and directory names that are difficult to distinguish from surrounding text, system messages, text that the user must type, and values for arguments or command options are in monospace.

Operating system differences

This book uses the UNIX[®] convention for specifying environment variables and for directory notation. When you are using the Windows[®] command line, replace *\$variable* with *%variable%* for environment variables and replace each forward slash (/) with a backslash (\) in directory paths. If you are using the bash shell on a Windows system, you can use the UNIX conventions.

Chapter 1. IBM Directory C-Client SDK overview

The Lightweight Directory Access Protocol (LDAP) provides TCP/IP access to LDAP-compliant servers. The IBM Tivoli 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 sections for more information:

- “LDAP version support”
- “LDAP API overview”
- “Updates for IBM Tivoli Directory Server C-Client Version 6.0” on page 3

LDAP version support

The IBM Tivoli 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, DN's 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 B, “LDAP V3 schema,” on page 197 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 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.

Source code, example makefile, and executable programs are provided for performing 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 either `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

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 Tivoli Directory Server C-Client Version 6.0

The following are enhancements available with the IBM Tivoli Directory Server C-Client Version 6.0.

New client utilities

The following client utilities have been added:

idsldaptrace

The administration tracing utility, idsldaptrace, is used to dynamically activate or deactivate tracing of the Directory Server. This extended operation can also be used to set the message level and specify the name of the file to the output is written.

New APIs

There are no new APIs for this release.

Kerberos

For IBM Tivoli Directory Server Version 6.0, Kerberos 1.4 is used on the AIX® operating system. For IBM Tivoli Directory Server Version 6.0, Kerberos 1.4.0.2 is used on the Windows operating system.

IBM Tivoli Directory Server Version 6.0 does not support Kerberos authentication on the Solaris, Linux® or HP operating systems.

Chapter 2. LDAP utilities

This chapter provides detailed information about the following client utilities:

- “`idsldapmodify`, `ldapmodify`, `idsldapadd`, `ldapadd`”
- “`idsldapchangepwd`, `ldapchangepwd`” on page 11
- “`idsldapdelete`, `ldapdelete`” on page 14
- “`idsldapexop`, `ldapexop`” on page 27
- “`idsldapmodrdn`, `ldapmodrdn`” on page 37
- “`idsldapsearch`, `ldapsearch`” on page 41
- “`idsldaptrace`, `ldaptrace`” on page 50
- “`ibmdirctl`, `idsdirctl`” on page 54

Note: You can change the source code for these LDAP utilities and build your own version of these LDAP utilities. However, any altered versions of these LDAP utilities will not be supported.

`idsldapmodify`, `ldapmodify`, `idsldapadd`, `ldapadd`

The LDAP modify-entry and LDAP add-entry tools

Synopsis

```
idsldapmodify | ldapmodify [-a] [-b] [-c] [-C charset] [-d debuglevel] [-D binddn]
                 [-e errorfile] [-f file] [-g] [-G realm] [-h ldaphost] [-i file]
                 [-k] [-K keyfile] [-l] [-m mechanism] [-M] [-n] [-N certificatename]
                 [-O maxhops] [-p ldapport] [-P keyfilepw] [-r] [-R] [-U username]
                 [-v] [-V] [-w passwd | ?] [-x] [-y proxydn] [-Y] [-Z]
```

```
idsldapadd | ldapadd [-a] [-b] [-c] [-C charset] [-d debuglevel] [-D binddn]
                 [-e errorfile] [-f file] [-g] [-G realm] [-h ldaphost] [-i file]
                 [-k] [-K keyfile] [-l] [-m mechanism] [-M] [-n] [-N certificatename]
                 [-O maxhops] [-p ldapport] [-P keyfilepw] [-r] [-R] [-U username]
                 [-v] [-V] [-w passwd | ?] [-x] [-y proxydn] [-Y] [-Z]
```

Description

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

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

To display syntax help for **idsldapmodify** or **idsldapadd**, type

```
idsldapmodify -?
```

or

```
idsldapadd -?
```

Options

- a** Add new entries. The default action for **idsldapmodify** is to modify existing entries. If invoked as **idsldapadd**, this flag is always set.
- b** Assume that any values that start with a ``/'` are binary values and that the actual value is in a file whose path is specified in place of the value.
- c** Continuous operation mode. Errors are reported, but **idsldapmodify** continues with modifications. Otherwise the default action is to exit after reporting an error.
- C charset**
Specifies that strings supplied as input to the **idsldapmodify** and **idsldapadd** utilities are represented in a local character set as specified by *charset*, and must be converted to UTF-8. When the **idsldapmodify** and **idsldapadd** 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 206 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>**
Sets the LDAP debugging level to *<debuglevel>*. This option causes the utility to generate debug output to stdout. The *<debuglevel>* is a bit mask that controls which output is generated with values up to 65535. This parameter is for use by IBM service personnel. See "Debugging levels" in the *IBM Tivoli Directory Server Version 6.0 Administration Guide* for additional information on debug levels.
- D binddn**
Use *binddn* to bind to the LDAP directory. *binddn* is a string-represented DN. When used with **-m DIGEST-MD5**, it specifies the authorization ID. It can be either a DN or an authzId string that starts with "u:" or "dn:".

Note: **-D binddn -w password** does not call bind functions on superuser DNs.
- e <errorfile>**
Specifies the file to which rejected entries are written. This option requires the **-c** continuous operation option. If the processing of an entry fails, that entry is written to the reject file and the count of rejected entries is increased. If the input to the **idsldapmodify** or **idsldapadd** command is from a file, when the file has been processed, the number of total entries written to the reject file is given.
- 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.

Note: This option is deprecated but still supported.
- g** Specifies not to strip the trailing spaces on attribute values.
- G realm**
Specify the name of the realm. When used with the **-m DIGEST-MD5**, the value is passed to the server during the bind.

-h *ldaphost*

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

-i *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.

-k Specifies to use server administration control.

This option sends the Server administration control. See “Server administration control” on page 258.

-K *keyfile*

Specify the name of the SSL or TLS 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 will first look 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 will be used, if present.

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

- AIX operating systems - `/opt/IBM/ldap/V6.0`
- HP-UX operating systems - `/opt/IBM/ldap/V6.0`
- Linux operating systems - `/opt/ibm/ldap/V6.0`
- Solaris operating systems - `/opt/IBM/ldap/V6.0`
- Windows operating systems - `<local_drive>:\Program Files\IBM\LDAP\V6.0` (This is the default install location. The actual `IDS_LDAP_HOME` is determined during installation.)

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 certificate authorities (CAs) that are trusted by the client. These types of X.509 certificates are also known as trusted roots. For more information on managing an SSL or TLS key database, see Chapter 4, “Using gsk7IKM,” on page 171. Also see the “Security functions” on page 11 and “LDAP_SSL” on page 152 for more information about SSL and certificates.

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

-l Do not replicate the entry.

This option sends the Do not replicate control. See “Do not replicate control” on page 248.

-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 Specify the no operation option to enable you to preview the result of the command you are issuing without actually performing the action on the directory. The changes that would be made are preceded by an exclamation mark and printed to standard output. Any syntax errors that are found in

the processing of the input file, before the calling of the functions that perform the changes to the directory, are displayed to standard error. This option is especially useful with the **-v** option for debugging operations, if errors are encountered.

-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 *maxhops*

Specify *maxhops* 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.

-U *username*

Specifies the username. This is required with **-m** DIGEST-MD5 and ignored when any other mechanism is used. The value *username* depends on what attribute the server is configured to use. It might be a uid or any other value that is used to locate the entry.

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

-V Specifies the LDAP version to be used by **idsldapmodify** 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 **idsldapmodify**, selects LDAP V3 as the preferred protocol by using `ldap_init` instead of `ldap_open`.

-w *password* | ?

Use *password* as the password for authentication. Use the ? to generate a password prompt. Using this prompt prevents your password from being visible through the **ps** command.

-x Use FIPS mode processing (SSL/TLS only).

-y *proxydn*

Specifies the DN to be used for proxied authorization.

- Y Use a secure TLS connection to communicate with the LDAP server. The -Y option is only supported when IBM's GSKit, is installed.
- Z Use a secure SSL connection to communicate with the LDAP server. The -Z option is only supported when the SSL component entry, as provided by IBM's GSKit, is installed.

Input format

The contents of file (or standard input if no `-i` flag is given on the command line) should conform to the LDIF format.

Alternative input format

An alternative input format is supported for compatibility with older versions of `idsldapmodify`. 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 `'\'` to continue values across lines and preserve new lines in the value itself.

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

`attr` should 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:

```
idsldapmodify -b -r -i /tmp/entrymods
```

will replace 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 `idsldapmodify` input 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:

```
idsldapmodify -b -r -i /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:

```
idsldapadd -i /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:

```
idsldapmodify -i /tmp/entrymods
```

removes John Doe's entry.

Notes

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

Security functions

To use the SSL or TLS-related functions associated with this utility, see “SSL, TLS notes” on page 55.

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.

See also

idsldapchangepwd, **idsldapdelete**, **idsldapexop**, **idsldapmodrdn**, **idsldapsearch**

idsldapchangepwd, ldapchangepwd

The LDAP modify password tool.

Synopsis

```
idsldapchangepwd | ldapchangepwd -D binddn -w passwd | ? -n newpassword | ?  
[-C charset] [-d debuglevel] [-G realm] [-h ldaphost]  
[-K keyfile] [-m mechanism] [-M] [-N certificatename]  
[-O maxhops] [-p ldapport] [-P keyfilepw] [-R]  
[-U username] [-v] [-V version] [-x] [-y proxydn] [-Y] [-Z] [-?]
```

Description

Sends modify password requests to an LDAP server.

Notes:

1. **idsldapchangepwd** cannot be used to change the administrator password or member of administrative group passwords. **idsldapchangepwd** works only with directory entries.
2. **idsldapchangepwd** works only on the **userpassword** attribute.

Options

-C *charset*

Specifies that the DN's supplied as input to the **idsldapchangepwd** 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 206 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>*

Sets the LDAP debugging level to *<debuglevel>*. This option causes the utility to generate debug output to stdout. The *<debuglevel>* is a bit mask that controls which output is generated with values up to 65535. This parameter is for use by IBM service personnel. See “Debugging levels” in

the *IBM Tivoli Directory Server Version 6.0 Administration Guide* for additional information on debug levels.

-D *binddn*

Use *binddn* to bind to the LDAP directory. *binddn* is a string-represented DN. When used with `-m DIGEST-MD5`, it specifies the authorization ID. It can be either a DN or an authzId string that starts with "u:" or "dn:".

-G *realm*

Specify the name of the realm. When used with the `-m DIGEST-MD5`, the value is passed to the server during the bind.

-h *ldaphost*

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

-K *keyfile*

Specify the name of the SSL or TLS 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 will first look 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 will be used, if present.

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

- AIX operating systems - `/opt/IBM/ldap/V6.0`
- HP-UX operating systems - `/opt/IBM/ldap/V6.0`
- Linux operating systems - `/opt/ibm/ldap/V6.0`
- Solaris operating systems - `/opt/IBM/ldap/V6.0`
- Windows operating systems - `<local_drive>:\Program Files\IBM\LDAP\V6.0` (This is the default install location. The actual `IDS_LDAP_HOME` is determined during installation.)

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 certificate authorities (CAs) that are trusted by the client. These types of X.509 certificates are also known as trusted roots. For more information on managing an SSL or TLS key database, see Chapter 4, "Using gsk7IKM," on page 171. Also see the "Security functions" on page 14 and "LDAP_SSL" on page 152 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 will be 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 *newpassword* | ?

Specifies the new password. Use the ? to generate a password prompt. Using this prompt prevents your password from being visible through the `ps` command.

- 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 *maxhops*
Specify *maxhops* to set the maximum number of hops that the client library takes when chasing referrals. The default hopcount is 10.
- p *ldappport*
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 may 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.
- U *username*
Specifies the username. This is required with -m DIGEST-MD5 and ignored when any other mechanism is used. The value *username* depends on what attribute the server is configured to use. It might be a uid or any other value that is used to locate the entry.
- v
Use verbose mode, with many diagnostics written to standard output.
- V *version*
Specifies the LDAP version to be used by **ldapdchangepwd** 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 **ldapdchangepwd**, selects LDAP V3 as the preferred protocol by using `ldap_init` instead of `ldap_open`.
- w *password* | ?
Use *password* as the password for authentication. Use the ? to generate a password prompt. Using this prompt prevents your password from being visible through the **ps** command.
- x
Use FIPS mode processing (SSL/TLS only).
- y *proxydn*
Specifies the DN to be used for proxied authorization.
- Y
Use a secure TLS connection to communicate with the LDAP server. The -Y option is only supported when IBM's GSKit, is installed.
- Z
Use a secure SSL connection to communicate with the LDAP server. The -Z option is only supported when the SSL component entry, as provided by IBM's GSKit, is installed.

-? Displays the syntax format.

Examples

The following command,

```
idsldapchangepwd -D "cn=John Doe" -w a1b2c3d4 -n wxyz9876
```

changes the password for the entry named with commonName "John Doe" from a1b2c3d4 to wxyz9876

Security functions

To use the SSL or TLS -related functions associated with this utility, see "SSL, TLS notes" on page 55.

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.

See also

idsldapadd, idsldapdelete, idsldapexop, idsldapmodify, idsldapmodrdn, idsldapsearch

idsldapdelete, ldapdelete

The LDAP delete-entry tool

Synopsis

```
idsldapdelete | ldapdelete [-c] [-C charset] [-d debuglevel] [-D binddn]
[-f file] [-G realm] [-h ldaphost] [-i file] [-k]
[-K keyfile] [-l] [-m mechanism] [-M] [-n]
[-N certificatename] [-O maxops] [-p ldapport]
[-P keyfilepw] [-R] [-s] [-U username} [-v] [-V version]
[-w passwd | ?] [-x] [-y proxydn] [-Y] [-Z] [dn]...
```

Description

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

idsldapdelete 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 DN's are deleted. Each DN is a string-represented DN. If no DN arguments are provided, a list of DN's is read from standard input, or from file if the **-i** or **-f** flag is used.

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

```
idsldapdelete -?
```

Options

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

-C *charset*

Specifies that the DN's supplied as input to the **idsldapdelete** 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 206 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>*

Sets the LDAP debugging level to *<debuglevel>*. This option causes the utility to generate debug output to stdout. The *<debuglevel>* is a bit mask that controls which output is generated with values up to 65535. This parameter is for use by IBM service personnel. See "Debugging levels" in the *IBM Tivoli Directory Server Version 6.0 Administration Guide* for additional information on debug levels.

-D *binddn*

Use *binddn* to bind to the LDAP directory. *binddn* is a string-represented DN. When used with -m DIGEST-MD5, it specifies the authorization ID. It can be either a DN or an authzId string that starts with "u:" or "dn:".

-f *file*

Read a series of lines from file, performing one LDAP delete for each line in the file. Each line in the file should contain a single distinguished name.

-G *realm*

Specify the name of the realm. When used with the -m DIGEST-MD5, the value is passed to the server during the bind.

-h *ldaphost*

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

-i *file*

Read a series of lines from file, performing one LDAP delete for each line in the file. Each line in the file should contain a single distinguished name.

-k

Specifies to use server administration control.

This option sends the Server administration control. See "Server administration control" on page 258.

-K *keyfile*

Specify the name of the SSL or TLS 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 will first look 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 will be used, if present.

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

- AIX operating systems - */opt/IBM/ldap/V6.0*
- HP-UX operating systems - */opt/IBM/ldap/V6.0*
- Linux operating systems - */opt/ibm/ldap/V6.0*
- Solaris operating systems - */opt/IBM/ldap/V6.0*
- Windows operating systems - *<local_drive>:\Program Files\IBM\LDAP\V6.0* (This is the default install location. The actual *IDS_LDAP_HOME* is determined during installation.)

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 certificate authorities (CAs) that are trusted by the client. These types of X.509 certificates are also known as trusted roots. For more information on managing an SSL or TLS key database, see Chapter 4, “Using gsk7IKM,” on page 171. Also see the “Security functions” on page 17 and “LDAP_SSL” on page 152 for more information about SSL and certificates.

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

-l Do not replicate the entry.

This option sends the Do not replicate control. See “Do not replicate control” on page 248.

-m *mechanism*

Use *mechanism* to specify the SASL mechanism to be used to bind to the server. The `ldap_sasl_bind_s()` API will be 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 Show what would be done, but don't actually modify entries. Useful for debugging in conjunction with **-v**.

-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 *maxhops*

Specify *maxhops* 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 may 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.

-s Use this option to delete the subtree rooted at the specified entry.

This option sends the Subtree delete control. See “Subtree delete control” on page 260.

-U *username*

Specifies the username. This is required with **-m** DIGEST-MD5 and ignored

when any other mechanism is used. The value *username* depends on what attribute the server is configured to use. It might be a uid or any other value that is used to locate the entry.

- v Use verbose mode, with many diagnostics written to standard output.
- V Specifies the LDAP version to be used by **idsldapdelete** 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 **idsldapdelete**, selects LDAP V3 as the preferred protocol by using `ldap_init` instead of `ldap_open`.
- w *password* | ? Use *password* as the password for authentication. Use the ? to generate a password prompt. Using this prompt prevents your password from being visible through the **ps** command.
- x Use FIPS mode processing (SSL/TLS only).
- y *proxydn* Specifies the DN to be used for proxied authorization.
- Y Use a secure TLS connection to communicate with the LDAP server. The **-Y** option is only supported when IBM's GSKit, is installed.
- Z Use a secure SSL connection to communicate with the LDAP server. The **-Z** option is only supported when the SSL component entry, as provided by IBM's GSKit, is installed.
- dn Specifies one or more DN arguments. Each DN should be a string-represented DN.

Examples

The following command,

```
idsldapdelete "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 *password* for deletion to be allowed (see the **-D** and **-w** options).

Notes

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

Security functions

To use the SSL or TLS -related functions associated with this utility, see "SSL, TLS notes" on page 55.

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.

See also

`idsldapadd`, `idsldapchangepwd`, `idsldapexop`, `idsldapmodify`, `idsldapmodrdn`, `idsldapsearch`

idsldapdiff, ldapdiff

The **idsldapdiff** utility identifies differences in a replica server and its master, and can be used to synchronize replicas.

Synopsis

To compare and optionally fix:

```
idsldapdiff | ldapdiff -b baseDN -sh host -ch host [-a] [-C countnumber]
[-cD dn] [-cK keyStore] [-cw password] [-cN keyStoreType]
[-cp port] [-cP keyStorePwd] [-ct trustStoreType]
[-cT trustStore] [-cY trustStorePwd] [-cZ] [-F] [-j]
[-L filename] [-O] [-sD dn] [-sK keyStore] [-sw password]
[-sN keyStoreType] [-sp port] [-sP keyStorePwd]
[-st trustStoreType] [-sT trustStore] [-sY trustStorePwd]
[-sZ]
```

or to compare schema only:

```
idsldapdiff | ldapdiff -S -sh host -ch host [-a] [-C countnumber]
[-cD dn] [-cK keyStore] [-cw password] [-cN keyStoreType]
[-cp port] [-cP keyStorePwd] [-ct trustStoreType]
[-cT trustStore] [-cY trustStorePwd] [-cZ] [-j]
[-L filename] [-O] [-sD dn] [-sK keyStore] [-sw password]
[-sN keyStoreType] [-sp port] [-sP keyStorePwd]
[-st trustStoreType] [-sT trustStore] [-sY trustStorePwd]
[-sZ]
```

Note: On UNIX systems, you must run the `idsldapdiff` command with a fully qualified path name, for example:

```
/opt/IBM/LDAP/V6.0/bin/idsldapdiff
```

Or you can also do one of the following:

- Set `/opt/IBM/LDAP/V6.0/bin` in the `$PATH` environment variable.
- Create a link manually. Use the following command:

```
ln -s /opt/IBM/LDAP/V6.0/bin/idsldapdiff /usr/bin/idsldapdiff
```

Description

The **idsldapdiff** command line utility is designed to compare two directory subtrees on two different directory servers to determine if their contents match. The utility can also optionally synchronize any entries that do not match. The following are two types of differences that might have to be synchronized:

- Entries that have the same DN, but different contents
- Entries that are present on one server, but not the other

The following is a list of operational attributes that `idsldapdiff` compares and fixes:

ACL related

- `aclEntry`
- `aclPropagate`
- `aclSource`
- `entryOwner`
- `ownerPropagate`
- `ownerSource`
- `ibm-filterAclEntry`
- `ibm-filterAclInherit`

Password Policy related

- pwdChangedTime
- pwdReset
- ibm-pwdAccountLocked

Other operational attributes

- ibm-entryUuid
- creatorsName
- createTimeStamp
- modifiersName
- modifyTimeStamp

Run the utility when no updates are being made to either of the directory servers. The administrator needs to quiesce or suspend all update activity to the two subtrees being compared. This must be done manually before invoking the compare tool. If the tool is run while updates are being made, it cannot be guaranteed that all discrepancies are accurately reported or fixed.

Note: The tool does not check on startup whether the servers are quiesced. When the tool is used in compare-only mode, the administrator might want to track down a small number of discrepancies as an alternative to stopping updates completely.

Use the tool with the server administration control (**-a** flag), if the fix operation is requested. The server administration control allows the tool to write to a read-only replica, and it also allows it to modify operational attributes such as `ibm-entryUuid`.

The **idsldapdiff** utility can be used to bring a master and replica server in sync before starting replication. The tool requires that the base DN, which is being compared, exists on both servers. If the base DN does not exist on either of the two servers, the utility gives an error and exits.

The tool traverses each entry in the directory subtree on the supplier server and compares its contents with the corresponding entry on the consumer server. Because information about each entry needs to be read, running the utility can take a long time and can generate lots of read requests to the supplier and consumer servers. Depending on how many differences are found and whether the fix operation is specified, the utility can also generate an equal amount of write requests to the consumer server.

Ideally, the tool is used only once between servers, when replication is initially setup. For example, if your topology has two peer masters and two replica servers, you might want to run **idsldapdiff** between peer 1 and peer 2. Then, if replication is suspended, run **idsldapdiff** concurrently between peer 1 and replica 1 and between peer 2 and replica 2. If replication is set up correctly, every change to the directory on the master servers is propagated to the replicas. However, if a problem occurs, the tool can be run to identify and correct replication problems. This utility is a diagnostic and corrective tool, it is not designed to run as routine maintenance. Depending on the replication-related errors observed in the log files, an administrator might decide to run the utility.

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

```
idsldapdiff -?
```

Note: The `idsldapdiff` utility displays a message after it has finished comparing every 100th entry.

Encryption considerations

`idsldapdiff` performs "cn=configuration" searches to determine the encryption settings on the server. Also, for performing searches and fixes, the administrator DN or administrator group DN is required. The tool fails if a bind DN other than the administrator DN or an administrative group member DN is used. Global administrators cannot run the `idsldapdiff compare` and `fix` options. Only administrators and administrator group members can run the `idsldapdiff compare` and `fix` options.

The supplier and consumer servers can have different encryption settings:

- Non-matching one-way
- Two-way and one-way
- Two-way with different stash files

Based on the types of encryption used, different behaviors occur when a password or any other password attribute is encountered.

Non-matching one-way

In this case the servers are using different types of one-way encryption. For example, the master server uses sha and the replica server uses crypt. The consumer values are directly overwritten with the value on the supplier. Running the `idsldapdiff` tool a second time on the same entries does not show any difference.

Two-way and one-way

In this case the one of the servers is using a two-way encryption algorithm like AES and the other server is using one-way encryption such as sha. Depending on whether the master server is using two-way or one-way encryption the behavior results are different. In this situation the performance of the `idsldapdiff` utility is degraded.

- When the supplier has a two-way encryption and the consumer has a one-way encryption, the `idsldapdiff` utility shows the two entries as always being different even if the actual values are the same. The supplier value is in plain text (decrypted because it is two-way) and consumer value is encrypted (because it is one way). Running the `idsldapdiff` tool a second time on the same entries still shows a difference even though the actual values are the same.
- When the supplier has a one-way encryption and the consumer has a two-way encryption, the consumer values are directly overwritten with the value on the supplier. Running the `idsldapdiff` tool a second time on the same entries does not show any difference.

Two-way encrypted data with different key stash files

In this case both servers are using two-way encryption but their stash files are generated with different seed or salt values. Because both servers perform decryption, performance of the `idsldapdiff` utility is degraded. If the plain text decrypted values are different, the synchronization process further degrades the performance of the `idsldapdiff` tool.

Notes:

1. The password policy attributes are synchronized by the `idsldapdiff` utility only if password policy is enabled on both of the servers.

2. The **idsldapdiff** utility checks the encryption settings on both of the servers and displays warning messages if the encryption settings are different both of the servers, or if the seed and salt values are different on both servers.
3. Use the **idsldapdiff** tool only for schema comparison. Do not use **idsldapdiff** with the **-F** option.

Options

The following options apply to the **idsldapdiff** command. There are two subgroupings that apply specifically to either the supplier server or the consumer server.

- a** Specifies inclusion of server administration control for writing to a read-only replica.
- b** *baseDN*
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** *countnumber*
Counts the number of non-matching entries. If more than the specified number of mismatches are found, the tool exits.
- F** This is the fix option. If specified, content on the consumer replica is modified to match the content of the supplier server. This cannot be used if the **-S** is also specified.
- j** Indicates to not include the following operational attributes in the LDIF file:
 - creatorsName
 - createTimeStamp
 - modifiersName
 - modifyTimeStamp

Note: The **-j** option is only valid when the **-L** option is specified.

- L** *<filename>*
If the **-F** option is not specified, use this option to generate an LDIF file for output. The LDIF file can be used to update the consumer to eliminate the differences.
- O** Displays DNs only for non-matching entries.

Note: This option overrides the **-F** and **-L** options.
- S** Specifies to compare the schema on both of the servers. Compares and fixes using **-S** can be made with any bind DN.
- x** Ignore extra entries on the consumer.

idsldapdiff performs two passes to make the servers are in sync. In the first pass, **idsldapdiff** traverses the Supplier server and does the following:

- Adds any extra entries on the supplier and to the consumer
- Compares and fixes entries that exist on both the servers

In the second pass, **idsldapdiff** traverses the Consumer to check for any extra entries on the Consumer. Specifying the **-x** option causes **idsldapdiff** to skip the second pass.

Options for a replication supplier

The following options apply to the supplier server and are denoted by an initial 's' in the option name.

-sD *dn* Use *dn* to bind to the LDAP directory. *dn* is a string-represented DN.

-sh *host*
Specifies the host name.

-sK *keyStore*
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 will first look 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 will be used, if present.

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

- AIX operating systems - `/opt/IBM/ldap/V6.0`
- HP-UX operating systems - `/opt/IBM/ldap/V6.0`
- Linux operating systems - `/opt/ibm/ldap/V6.0`
- Solaris operating systems - `/opt/IBM/ldap/V6.0`
- Windows operating systems - `<local_drive>:\Program Files\IBM\LDAP\V6.0` (This is the default install location. The actual `IDS_LDAP_HOME` is determined during installation.)

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 certificate authorities (CAs) that are trusted by the client. These types of X.509 certificates are also known as trusted roots. For more information on managing an SSL key database, see Chapter 4, "Using gsk7IKM," on page 171. Also see the "Security functions" on page 27 and "LDAP_SSL" on page 152 for more information about SSL and certificates.

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

-sN *keyStoreType*
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. *keyStoreType* is not required if a default certificate/private key pair has been designated as the default. Similarly, *keyStoreType* is not required if there is a single certificate/private key pair in the designated key database file. This parameter is ignored if neither **-sZ** nor **-sK** is specified.

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

-sP *keyStorePwd*

Specify the key database password. This password is required to access the encrypted information in the key database file, which may 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 **-sP** parameter is not required. This parameter is ignored if neither **-sZ** nor **-sK** is specified.

-st *trustStoreType*

Specify the label associated with the client certificate in the trust 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. *trustStoreType* is not required if a default certificate/private key pair has been designated as the default. Similarly, *trustStoreType* is not required if there is a single certificate/private key pair in the designated key database file. This parameter is ignored if neither **-sZ** nor **-sT** is specified.

-sT *trustStore*

Specify the name of the SSL trust database file with default extension of **tdb**. If the trust database file is not in the current directory, specify the fully-qualified trust database filename. If a trust database filename is not specified, this utility will first look 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 will be used, if present.

A default keyring file that is, `ldapkey.tdb`, and the associated password stash file that is, `ldapkey.sth`, are installed in the `/etc` directory under `IDS_LDAP_HOME`, where `IDS_LDAP_HOME` is the path to the installed LDAP support. `IDS_LDAP_HOME` varies by operating system platform:

- AIX operating systems - `/opt/IBM/ldap/V6.0`
- HP-UX operating systems - `/opt/IBM/ldap/V6.0`
- Linux operating systems - `/opt/ibm/ldap/V6.0`
- Solaris operating systems - `/opt/IBM/ldap/V6.0`
- Windows operating systems - `<local_drive>:\Program Files\IBM\LDAP\V6.0` (This is the default install location. The actual `IDS_LDAP_HOME` is determined during installation.)

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 certificate authorities (CAs) that are trusted by the client. These types of X.509 certificates are also known as trusted roots. For more information on managing an SSL key database, see Chapter 4, "Using gsk7IKM," on page 171. Also see the "Security functions" on page 27 and "LDAP_SSL" on page 152 for more information about SSL and certificates.

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

-sw *password* | ?

Use *password* as the password for authentication. Use the ? to generate a password prompt. Using this prompt prevents your password from being visible through the **ps** command.

-sY The password for the trusted database.

-sZ Use a secure SSL connection to communicate with the LDAP server. The **-Z** option is only supported when the SSL component entry, as provided by IBM's GSKit, is installed.

Options for a replication consumer

The following options apply to the consumer server and are denoted by an initial 'c' in the option name.

-cD dn Use *dn* to bind to the LDAP directory. *dn* is a string-represented DN.

-ch host

Specifies the host name.

-cK keyStore

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 will first look 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 will be used, if present.

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

- AIX operating systems - `/opt/IBM/ldap/V6.0`
- HP-UX operating systems - `/opt/IBM/ldap/V6.0`
- Linux operating systems - `/opt/ibm/ldap/V6.0`
- Solaris operating systems - `/opt/IBM/ldap/V6.0`
- Windows operating systems - `<local_drive>:\Program Files\IBM\LDAP\V6.0` (This is the default install location. The actual `IDS_LDAP_HOME` is determined during installation.)

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 certificate authorities (CAs) that are trusted by the client. These types of X.509 certificates are also known as trusted roots. For more information on managing an SSL key database, see Chapter 4, "Using gsk7IKM," on page 171. Also see the "Security functions" on page 27 and "LDAP_SSL" on page 152 for more information about SSL and certificates.

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

-cN keyStoreType

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. *keyStoreType* is not required if a default certificate/private key pair has been designated as the default. Similarly, *keyStoreType* is not required if there is a single certificate/private key pair in the designated key database file. This parameter is ignored if neither **-cZ** nor **-cK** is specified.

-cp *ldapport*

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

-cP *keyStorePwd*

Specify the key database password. This password is required to access the encrypted information in the key database file, which may 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 **-cP** parameter is not required. This parameter is ignored if neither **-cZ** nor **-cK** is specified.

-ct *trustStoreType*

Specify the label associated with the client certificate in the trust 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. *trustStoreType* is not required if a default certificate/private key pair has been designated as the default. Similarly, *trustStoreType* is not required if there is a single certificate/private key pair in the designated key database file. This parameter is ignored if neither **-cZ** nor **-cT** is specified.

-cT *trustStore*

Specify the name of the SSL trust database file with default extension of **tdb**. If the trust database file is not in the current directory, specify the fully-qualified trust database filename. If a trust database filename is not specified, this utility will first look 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 will be used, if present.

A default keyring file that is, *ldapkey.tdb*, and the associated password stash file that is, *ldapkey.sth*, are installed in the */etc* directory under *IDS_LDAP_HOME*, where *IDS_LDAP_HOME* is the path to the installed LDAP support. *IDS_LDAP_HOME* varies by operating system platform:

- AIX operating systems - */opt/IBM/ldap/V6.0*
- HP-UX operating systems - */opt/IBM/ldap/V6.0*
- Linux operating systems - */opt/ibm/ldap/V6.0*
- Solaris operating systems - */opt/IBM/ldap/V6.0*
- Windows operating systems - *<local_drive>:\Program Files\IBM\LDAP\V6.0* (This is the default install location. The actual *IDS_LDAP_HOME* is determined during installation.)

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 certificate authorities (CAs) that are trusted by the client. These types of X.509 certificates are also known as trusted roots. For more information on managing an SSL key database, see Chapter 4, "Using gsk7IKM," on page 171. Also see the "Security functions" on page 27 and "LDAP_SSL" on page 152 for more information about SSL and certificates.

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

-cw *password* | ?

Use *password* as the password for authentication. Use the ? to generate a password prompt. Using this prompt prevents your password from being visible through the **ps** command.

-cY The password for the trusted database.

-cZ Use a secure SSL connection to communicate with the LDAP server. The **-cZ** option is only supported when the SSL component entry, as provided by IBM's GSKit, is installed.

Examples

```
idsldapdiff -b <baseDN> -sh <supplierhostname> -ch <consumerhostname> [options]
```

or

```
idsldapdiff -S -sh <supplierhostname> -ch <consumerhostname> [options]
```

As an illustration of how the utility works, set up two servers one as a master server and other as a replica server. Assume that Suffix *o=ibm, c=us* is present on both the servers. Create two LDIF files *master.ldif* and *replica.ldif*

master.ldif with entries

```
dn: cn=Entry1,o=ibm,c=us
objectclass: inetOrgPerson
objectclass: organizationalPerson
objectclass: person
objectclass: top
objectclass: ePerson
sn: entry1
cn: testEntry
```

```
dn: cn=Entry2,o=ibm,c=us
objectclass: inetOrgPerson
objectclass: organizationalPerson
objectclass: person
objectclass: top
objectclass: ePerson
sn: entry2
cn: testEntry
```

replica.ldif with entries

```
dn: cn=Entry2,o=ibm,c=uschangeType: add
objectclass: inetOrgPerson
objectclass: organizationalPerson
objectclass: person
objectclass: top
objectclass: ePerson
sn: abcd
cn: testEntry
```

```
dn: cn=Entry3,o=ibm,c=us
changeType: add
objectclass: inetOrgPerson
objectclass: organizationalPerson
objectclass: person
objectclass: top
objectclass: ePerson
sn: entry3
cn: testEntry
```

Run the **idsldapdiff** command:

```
idsldapdiff -b o=ibm,c=us -sh <master> -sD cn=root -sw <passwd> -ch <replica>
-cD cn=root -cw <passwd> -F -a
```


The resulting actions are:

1. Entry `cn=Entry1,o=ibm,c=us` gets added on Replica server. This entry is on the master server, but was not on the replica server.
2. Entry `cn=Entry2,o=ibm,c=us` gets modified on Replica server. The value of `sn` field gets modified to match the value on the master server.
3. Entry `cn=Entry3,o=ibm,c=us` get deleted from Replica server. This entry is extra on the replica server that was not on the master server.

Notes

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

Security functions

To use the SSL or TLS -related functions associated with this utility, see “SSL, TLS notes” on page 55.

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.

idsldapexop, ldapexop

The LDAP extended operation tool.

Synopsis

```
idsldapexop | ldapexop [-C charset] [-d debuglevel] [-D binddn] [-e] [-G realm]
[-h ldaphost] [-help] [-K keyfile] [-m mechanism] [-N certificatename]
[-p ldapport] [-P keyfilepw] [-?] [-U username] [-v] [-w passwd | ?]
[-x] [-y proxyDN] [-Y] [-Z]
-op {acctstatus | cascrepl | clearlog | controlqueue | controlrepl |
controlreplerr | evaluategroups | getattributes | getlogsize |
getusertype | quiesce | readconfig | readlog | repltopology |
stopsserver | unbind | uniqueattr }
```

Description

The `idsldapexop` utility is a command-line interface that provides the capability to bind to a directory and issue a single extended operation along with any data that makes up the extended operation value.

The `idsldapexop` utility supports the standard host, port, SSL, TLS, and authentication options used by all of the LDAP client utilities. In addition, a set of options is defined to specify the operation to be performed, and the arguments for each extended operation

To display syntax help for `idsldapexop`, type:

```
idsldapexop -?
```

or

```
idsldapexop -help
```

Options

The options for the `idsldapexop` command are divided into two categories:

1. General options that specify how to connect to the directory server. These options must be specified before operation specific options.
2. Extended operation option that identifies the extended operation to be performed.

General options

These options specify the methods of connecting to the server and must be specified before the **-op** option.

-C *<charset>*

Specifies that the DNs supplied as input to the **idsldapexop** 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 206 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>*

Sets the LDAP debugging level to *<debuglevel>*. This option causes the utility to generate debug output to stdout. The *<debuglevel>* is a bit mask that controls which output is generated with values up to 65535. This parameter is for use by IBM service personnel. See "Debugging levels" in the *IBM Tivoli Directory Server Version 6.0 Administration Guide* for additional information on debug levels.

-D *<binddn>*

Use *binddn* to bind to the LDAP directory. *binddn* is a string-represented DN. When used with **-m** DIGEST-MD5, it specifies the authorization ID. It can be either a DN or an authzId string that starts with "u:" or "dn:".

-e Displays the ldap library version information and then exits.

-G *<realm>*

Specify the name of the realm. When used with the **-m** DIGEST-MD5, the value is passed to the server during the bind.

-h *<ldaphost>*

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

-help Displays the usage

-K *<keyfile>*

Specify the name of the SSL or TLS 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 will first look 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 will be used, if present.

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

- AIX operating systems - */opt/IBM/ldap/V6.0*
- HP-UX operating systems - */opt/IBM/ldap/V6.0*
- Linux operating systems - */opt/ibm/ldap/V6.0*
- Solaris operating systems - */opt/IBM/ldap/V6.0*

- Windows operating systems - `<local_drive>:\Program Files\IBM\LDAP\V6.0` (This is the default install location. The actual `IDS_LDAP_HOME` is determined during installation.)

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 certificate authorities (CAs) that are trusted by the client. These types of X.509 certificates are also known as trusted roots. For more information on managing an SSL or TLS key database, see Chapter 4, "Using gsk7IKM," on page 171. Also see the "Security functions" on page 35 and "LDAP_SSL" on page 152 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 will be used. The `-m` parameter is ignored if `-V 2` is set. If `-m` is not specified, simple authentication is used.

-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.

-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 may 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.

-? Displays the syntax format.

-U *<username>*

Specifies the username. This is required with `-m DIGEST-MD5` and ignored when any other mechanism is used. The value *username* depends on what attribute the server is configured to use. It might be a uid or any other value that is used to locate the entry.

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

-w *<password>* | ?

Use *password* as the password for authentication. Use the ? to generate a password prompt. Using this prompt prevents your password from being visible through the `ps` command.

-x Use FIPS mode processing (SSL/TLS only).

- y <proxyDN>
Sets a proxied ID for proxied authorization operation.
- Y Use a secure TLS connection to communicate with the LDAP server. The -Y option is only supported when IBM's GSKit, is installed.
- Z Use a secure SSL connection to communicate with the LDAP server. The -Z option is only supported when the SSL component entry, as provided by IBM's GSKit, is installed.

Extended operations option

The **-op** extended-op option identifies the extended operation to be performed. The extended operation can be one of the following values:

- **acctStatus -d<userDN>**: password policy account status extended operation. This operation enables a directory administrator to query the server as to the account status of any entry that contains a userPassword attribute. The *userDN* is the DN of the user account that is being queried. The status for the account is open, locked, or expired.

Examples:

```
idsldapexop -op acctStatus -d cn=Bob Garcia,ou=austin,o=ibm,c=us
```

- **cascrepl -action<actionvalue> -rc<contextDN> [options]**: cascading control replication extended operation. The requested action is applied to the specified server and also passed along to all replicas of the given subtree. If any of these are forwarding replicas, they pass the extended operation along to their replicas. The operation cascades over the entire replication topology.

-action {quiesce | unquiesce | replnow | wait}

This is a required attribute that specifies the action to be performed.

quiesce

No further updates are allowed, except by replication.

unquiesce

Resume normal operation, client updates are accepted.

replnow

Replicate all queued changes to all replica servers as soon as possible, regardless of schedule.

wait

Wait for all updates to be replicated to all replicas.

-rc contextDn

This is a required attribute that specifies the root of the subtree.

options

-timeout secs

This is an optional attribute that if present, specifies the timeout period in seconds. If not present, or 0, the operation waits indefinitely.

Example:

```
idsldapexop -op cascrepl -action quiesce -rc "o=acme,c=us" -timeout 60
```

- **clearlog -log<logname>**: clear log file extended operation

-log {audit | bulkload | cli | slapd | idsdiradm | adminAudit | debug | LostAndFound | config}

This is a required attribute that specifies the log file to be cleared.

Example:

```
idsldapexop -D <bindDN> -W <password> -op clearlog -log audit
```

- **controlqueue -skip<skipvalue> -ra<agreementDN>**: control queue extended operation

-skip {all | change-id}

This is a required attribute.

- **all** indicates to skip all pending changes for this agreement.
- **change-id** identifies the single change to be skipped. If the server is not currently replicating this change, the request fails.

-ra agreementDN

This is a required attribute that specifies the DN of the replication agreement.

Examples:

```
idsldapexop -op controlqueue -skip all -ra "cn=server3,  
ibm-replicaSubentry=master1-id,ibm-replicaGroup=default,  
o=acme,c=us"
```

```
idsldapexop -op controlqueue -skip 2185 -ra "cn=server3,  
ibm-replicaSubentry=master1-id,ibm-replicaGroup=default,  
o=acme,c=us"
```

- **controlrepl -action<actionvalue> {-rc<contextDN> | -ra<agreementDN>}**: control replication extended operation

-action {suspend | resume | replnow}

This is a required attribute that specifies the action to be performed.

-rc contextDn | -ra agreementDn

The **-rc contextDn** is the DN of the replication context. The action is performed for all agreements for this context. The **-ra agreementDn** is the DN of the replication agreement. The action is performed for the specified replication agreement.

Example:

```
idsldapexop -op controlrepl -action suspend -ra "cn=server3,  
ibm-replicaSubentry=master1-id,ibm-replicaGroup=default,  
o=acme,c=us"
```

- **controlreplerr [-delete failure-ID | all] [-retry failure-ID | all] [-show failure-ID]**
-ra<agreementDN>: control replication error extended operation

-delete failure-ID | all

Specifies to remove the failed update, where

all Specifies to delete all the failed updates for this agreement.

failure-ID

Specifies to delete only the failed update specified by the failure-ID for this agreement.

-retry failure-ID | all

Specifies to retry the failed update, where

all Specifies to retry all the failed updates for this agreement.

failure-ID

Specifies to retry only the failed update specified by the failure-ID for this agreement.

-show failure-ID

Specifies to show the failed update specified by the failure-ID.

-ra *agreementDn*

The **-ra** *agreementDn* is the DN of the replication agreement. The action is performed for the specified replication agreement.

Example:

```
idsldapexop -op controlreplerr -delete all -ra "cn=server3,  
ibm-replicaSubentry=master1-id,ibm-replicaGroup=default,  
o=acme,c=us"
```

- **evaluategroups -d** *<specificDN>* [**-a** *attribute value pairs...*] : request evaluategroups extended operation

-d *<specificDN>*

Specifies the DN that is to be evaluated to determine what groups it belongs to.

-a *attribute value pairs...*

Specifies a list of whitespace-separated list of attribute value pairs. Each attribute value pair is in the attr=value format. If the **-a** option is not specified, the specified DN is evaluated for static groups only.

An attribute value pair is an attribute type and attribute value separated by an equal sign. A user's attributes are required for evaluating group membership for dynamic group. When the server receives an evaluate group request with attributes, it is these attributes that are used in the group evaluation.

Example:

```
idsldapexop -op evaluategroups -d "cn=John Smith,ou=Austin,o=ibm,c=us" -a  
departmentNumber=G8R
```

- **getattributes -attrType***<type>* **-matches** *<value>*

-attrType {operational | language_tag | attribute_cache | unique | configuration}

This is a required attribute that specifies type of attribute being requested.

-matches {true | false}

Specifies whether the list of attributes returned matches the attribute type specified by the **-attrType** option.

Example:

```
idsldapexop -op getattributes -attrType unique -matches true
```

Returns a list of all attributes that can be defined as unique attributes.

```
idsldapexop -op getattributes -attrType unique -matches false
```

Returns a list of all attributes that have been not been defined as unique attributes.

- **getlogsize -log***<logname>*: request log file size extended operation

-log {audit | bulkload | cli | slapd | idsdiradm | adminAudit | debug | LostAndFound | config}

This is a required attribute that specifies the log file to be queried. The size of the log file, in lines, is written to standard output.

Example:

```
idsldapexop -D <AdminDN> -w <Adminpw> -op getlogsize -log slapd  
2000 lines
```

- **getusertype**: request user type extended operation
This extended operation returns the user type based on the bound DN.

Example:

```
idsldapexop -D <AdminDN> -w <Adminpw> -op getusertype
```

returns:

```
User      : root_administrator
Role(s)   : server_config_administrator directory_administrator
```

See “User type and user roles for extended operations” on page 35 for more information.

- **quiesce -rc <contextDN>[options]**: quiesce or unquiesce subtree extended operation

-rc <contextDN>

This is a required attribute that specifies the DN of the replication context (subtree) to be quiesced or unquiesced.

options

-end This is an optional attribute that if present, specifies to unquiesce the subtree. If not specified the default is to quiesce the subtree.

Examples:

```
idsldapexop -op quiesce -rc "o=acme,c=us"
```

```
idsldapexop -op quiesce -end -rc "o=ibm,c=us"
```

- **readconfig -scope<scopevalue>**: reread configuration file extended operation

-scope {entire | single<entry DN><attribute> | entry <entry DN> | subtree <entry DN>}

This is a required attribute.

- **entire** indicates to reread the entire configuration file.
- **single <entry DN><attribute>** means to read the single entry and attribute specified.
- **entry <entry DN>** means to read the entry specified.
- **subtree <entry DN>** means to read the entry and the entire subtree under it.

Examples:

```
idsldapexop -D <AdminDN> -w <Adminpw> -op readconfig -scope entire
```

```
idsldapexop -D <AdminDN> -w <Adminpw> -op readconfig -scope
single "cn=configuration" ibm-slapdAdminPW
```

- **readlog -log <logname> -lines <value>**: request lines from log file extended operation

-log {audit | bulkload | cli | slapd | idsdiradm | adminAudit | debug | LostAndFound | config}

This is a required attribute that specifies the log file to be queried.

-lines {<first><last> | all}

This is a required attribute that specifies either the first and last lines to be read from the file or all lines. Lines are numbered starting at 0. The specified lines are written to standard output.

Examples:

```
idsldapexop -D <AdminDN> -w <Adminpw> -op readlog -log audit -lines 10 20
```

```
idsldapexop -op readlog -log slapd -lines all
```

- **repltopology -rc<contextDN> [options]**: replication topology extended operation. This operation replicates the replication topology related entries under the specified context.

-rc *contextDn*

This is a required attribute that specifies the root of the subtree.

options

-timeout *secs*

This is an optional attribute that if present, specifies the timeout period in seconds. If not present, or 0, the operation waits indefinitely.

-ra *agreementDn*

The **-ra** *agreementDn* is the DN of the replication agreement. The action is performed for the specified replication agreement. If the **-ra** option is not specified, the action is performed for all the replication agreements defined under the context.

Example:

```
idsldapexop -op repltopology -rc "o=acme,c=us" -ra "cn=server3,  
ibm-replicaSubentry=master1-id,ibm-replicaGroup=default,  
o=acme,c=us"-timeout 60
```

- **stopserver**: stop the IBM Tivoli Directory Server

Example:

```
idsldapexop -D <adminDn> -w <adminpw> -op stopserver
```

- **unbind {-dn<specificDN> | -ip<sourceIP> | -dn<specificDN> -ip<sourceIP> | all}**: disconnect connections based on DN, IP, DN/IP or disconnect all connections. All connections without any operations and all connections with operations on the work queue are ended immediately. If a worker is currently working on a connection, it is ended as soon as the worker completes that one operation.

-dn<specificDN>

Issues a request to end a connection by DN only. This request results in the purging of all the connections bound on the specified DN.

-ip<sourceIP>

Issues a request to end a connection by IP only. This request results in the purging of all the connections from the specified IP source.

-dn<specificDN> -ip<sourceIP>

Issues a request to end a connection determined by a DN/IP pair. This request results in the purging of all the connections bound on the specified DN and from the specified IP source.

-all

Issues a request to end all the connections. This request results in the purging of all the connections except the connection from where this request originated. This attribute cannot be used with the **-dn** or **-ip** attributes

Examples:

```
idsldapexop -D <AdminDN> -w <Adminpw> -op unbind -dn cn=john
```

```
idsldapexop -D <AdminDN> -w <Adminpw> -op unbind -ip 9.182.173.43
```



```
idsldapexop -D <AdminDN> -w <Adminpw> -op unbind -dn cn=john -ip 9.182.173.43
```

```
idsldapexop -D <AdminDN> -w <Adminpw> -op unbind -all
```

- **uniqueattr -a <attributeType>**: identify all nonunique values for a particular attribute.

-a <attribute>

Specify the attribute for which all conflicting values are listed.

Note: Duplicate values for binary, operational, configuration attributes, and the objectclass attribute are not displayed. These attributes are not supported extended operations for unique attributes.

Example:

```
idsldapexop -D <AdminDN> -w <Adminpw> -op uniqueattr -a "uid"
```

The following line is added to the configuration file under the "cn=Directory,cn=RDBM Backends,cn=IBM Directory,cn=Schema,cn=Configuration" entry for this extended operation.

```
ibm-slapdPlugin:extendedop /bin/libback-rdbm.dll initUniqueAttr
```

Notes

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

Security functions

To use the SSL or TLS -related functions associated with this utility, see "SSL, TLS notes" on page 55.

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.

User type and user roles for extended operations

The following are the users and their roles for extended operations.

Root administrator

An administrative user whose simple and External with SSL or TLS bind credentials are stored under the cn=Configuration entry. This user's Kerberos bind credentials (optional) are stored under the cn=Kerberos,cn=Configuration entry. This user's Digest-MD5 bind credentials (optional) are stored under the cn=Digest,cn=Configuration entry. In addition, this type of user can bind to the Admin Daemon.

Roles:

Server configuration administrator

This user has unrestricted access to all information in the configuration backend and can start/stop the server. The user can issue dynamic configuration updates.

Directory administrator

This user has unrestricted access to directory data outside the configuration backend (schema, and RDBM backends). This user can search for one or

two attributes in the configuration backend. This user may not have any authority to the operating specific backends (i5/OS system projection backend, z/OS™ RACF® SDBM).

Administrative group member

An administrative user whose simple, External with SSL or TLS , Kerberos (optional), and Digest-MD5 (optional) credentials are stored under an entry in the subtree cn=AdminGroup,cn=Configuration. In addition, this type of user can bind to the Admin Daemon.

Roles:

Server configuration group member

This user has access to all configuration information except the administrator and admin group credentials. This user has the ability to start and stop the server. The user does not have the ability to add or remove members from the administrative group. The user cannot modify the administration password policy. The user is not able to modify the DN, password, Kerberos ID, or Digest-MD5 ID of any administrative group member entry under cn=AdminGroup,cn=Configuration. If the user is an Administrative Group Member the user is able to modify his own password, but is not able to modify his own DN, Kerberos ID, or Digest-MD5 ID. This user is also not able to see the password of any other administrative group member or the IBM Tivoli Directory Server administrator.

Members of the administrative group can view the administration daemon audit log and settings but not modify them. Only the administrator is enabled to access, change or clear the administration daemon audit log files.

In addition, this user is not able to add, delete, or modify the audit log setting (the entire cn=Audit,cn=Configuration entry) or clear the audit log. The user is not able to add or delete the cn=Kerberos,cn=Configuration or cn=Digest,cn=Configuration entries, but is able to search all attributes under these entries. The user is able to modify all attributes under these entries except the Kerberos and Digest-MD5 root administrator bind attributes. These users are not able to search or modify the ibm-slapdAdminDN, ibm-slapdAdminGroupEnabled or ibm-slapdAdminPW attributes under the cn=Configuration entry. The user can issue dynamic configuration updates.

Directory administrator

This user has unrestricted access to directory data outside the configuration backend (schema, and RDBM backends). This user can search for one or two attributes in the configuration backend. This user may not have any authority to the operating specific backends (i5/OS system projection backend, z/OS RACF SDBM).

LDAP user type

A regular LDAP user whose credentials are stored in the DIT of the LDAP Server. The user's simple and external with SSL or TLS bind DN is the DN of an entry in the DIT. The user's password is stored in the userpassword attribute of this entry.

Roles:

LDAP User Role

A user having almost no access to the configuration backend. This user can

search for one or two attributes in the configuration backend. The user's access to directory data (schema, and RDBM backends) is controlled by ACLs.

Global administration group member

This user has his credentials stored in the same location as the "ldap_user_type" and has the same bind DN and password attribute settings. This user differs from the "ldap_user_type" in that he belongs to the global administration group entry that is stored in globalGroupName=globalAdminGroup,cn=ibmpolicies.

Roles:

Directory administrator

This user has unrestricted access to directory data outside the configuration backend (schema, and RDBM backends). This user may not have any authority to the operating specific backends (i5/OS system projection backend, z/OS RACF SDBM).

See also

idsldapadd, idsldapchangepwd, idsldapdelete, idsldapmodify, idsldapmodrdn, idsldapsearch

idsldapmodrdn, ldapmodrdn

The LDAP modify-entry RDN tool

Synopsis

```
idsldapmodrdn | ldapmodrdn [-c] [-C charset] [-d debuglevel][-D binddn]
[-f file] [-G realm] [-h ldaphost] [-i file] [-k] [-K keyfile] [-l]
[-m mechanism] [-M] [-n] [-N certificatename] [-O hopcount]
[-p ldapport] [-P keyfilepw] [-r] [-R] [-s newSuperior]
[-U username] [-v] [-V] [-w passwd | ?] [-x] [-y proxydn] [-Y]
[-Z] [dn newrdn | [-i file]]
```

Description

idsldapmodrdn is a command-line interface to the ldap_rename library call.

idsldapmodrdn opens a connection to an LDAP server, binds, modifies the RDN of an entry and can change the parent of the entry. The entry information is read from standard input, from a file through the use of the -i option, or from the command-line pair dn, rdn, or the newSuperior option.

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

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

```
idsldapmodrdn -?
```

Options

- c Continuous operation mode. Errors are reported, but **idsldapmodrdn** 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 **idsldapmodrdn** 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 206 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>*

Sets the LDAP debugging level to *<debuglevel>*. This option causes the utility to generate debug output to stdout. The *<debuglevel>* is a bit mask that controls which output is generated with values up to 65535. This parameter is for use by IBM service personnel. See "Debugging levels" in the *IBM Tivoli Directory Server Version 6.0 Administration Guide* for additional information on debug levels.

-D *binddn*

Use *binddn* to bind to the LDAP directory. *binddn* is a string-represented DN. When used with `-m DIGEST-MD5`, it specifies the authorization ID. It can be either a DN or an authzId string that starts with "u:" or "dn:".

-f *file* Read entry modification information from specified file.

-G *realm*

Specify the name of the realm. When used with the `-m DIGEST-MD5`, the value is passed to the server during the bind.

-h *ldaphost*

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

-i *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>`").

-k Specifies to use server administration control.

This option sends the Server administration control. See "Server administration control" on page 258.

-K *keyfile*

Specify the name of the SSL or TLS 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 will first look 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 will be used, if present.

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

- AIX operating systems - `/opt/IBM/ldap/V6.0`
- HP-UX operating systems - `/opt/IBM/ldap/V6.0`
- Linux operating systems - `/opt/ibm/ldap/V6.0`
- Solaris operating systems - `/opt/IBM/ldap/V6.0`
- Windows operating systems - `<local_drive>:\Program Files\IBM\LDAP\V6.0` (This is the default install location. The actual `IDS_LDAP_HOME` is determined during installation.)

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 certificate authorities (CAs) that are trusted by the client. These types of X.509 certificates are also known as trusted roots. For more information on managing an SSL or TLS key database, see Chapter 4, "Using gsk7IKM," on page 171. Also see the "Security functions" on page 41 and "LDAP_SSL" on page 152 for more information about SSL and certificates.

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

-l Do not replicate the entry.

This option sends the Do not replicate control. See "Do not replicate control" on page 248.

-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 Show what would be done, but don't actually modify entries. Useful for debugging in conjunction with **-v**.

-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 *ldappport*

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 may 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.

- s *newSuperior*
Specifies the DN of the new superior entry under which the renamed entry is relocated. The *newSuperior* argument may be the zero-length string (-s "").
 - U *username*
Specifies the username. This is required with -m DIGEST-MD5 and ignored when any other mechanism is used. The value *username* depends on what attribute the server is configured to use. It might be a uid or any other value that is used to locate the entry.
 - v
Use verbose mode, with many diagnostics written to standard output.
 - V
Specifies the LDAP version to be used by **idsldapmodrdn** 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 **idsldapmodrdn**, selects LDAP V3 as the preferred protocol by using `ldap_init` instead of `ldap_open`.
 - w *password* | ?
Use *password* as the password for authentication. Use the ? to generate a password prompt. Using this prompt prevents your password from being visible through the **ps** command.
 - x
Use FIPS mode processing (SSL/TLS only).
 - y *proxydn*
Specifies the DN to be used for proxied authorization.
 - Y
Use a secure TLS connection to communicate with the LDAP server. The -Y option is only supported when IBM's GSKit, is installed.
 - Z
Use a secure SSL connection to communicate with the LDAP server. The -Z option is only supported when the SSL component entry, as provided by IBM's GSKit, is installed.
- dn newrdn**
See the following section, "Input format for dn newrdn" for more information.

Input format for dn newrdn

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

Distinguished Name (DN)

Relative Distinguished Name (RDN)

One or more blank lines may be used to separate each DN 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:

```
idsldapmodrdn -r -i /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.

The command:

```
idsldapmodrdn -s "o=IBM,c=US" "cn=Modify Me,o=University of Life,c=US"
"cn=The New Me"
```

changes the RDN of the Modify Me entry from Modify Me to The New Me. The entry is moved from underneath the University of Life entry to underneath the IBM entry.

Notes

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

Security functions

To use the SSL or TLS -related functions associated with this utility, see “SSL, TLS notes” on page 55.

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.

See also

idsldapadd, idsldapchangepwd, idsldapdelete, idsldapexop, idsldapmodify, idsldapsearch

idsldapsearch, ldapsearch

The LDAP search tool and sample program

Synopsis

```
idsldapsearch | ldapsearch [-a deref] [-A] [-b searchbase] [-B] [-C charset]
[-d debuglevel] [-D binddn] [-f file] [-F sep] [-G realm] [-h ldaphost] [-i file]
[-k] [-K keyfile] [-l timelimit] [-L] [-m mechanism] [-M] [-n] [-N certificatename]
[-o attr_type] [-O maxhops] [-p ldapport] [-P keyfilepw] [-q pagesize]
[-R] [-s scope] [-t] [-T seconds] [-U username] [-v] [-V version]
[-w passwd | ?] [-x] [-y proxydn] [-Y] [-z sizelimit] [-Z]
filter [-9 p] [-9 s] [attrs...]
```

Description

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

idsldapsearch opens a connection to an LDAP server, binds, and performs a search using the filter. The filter should conform to the string representation for LDAP filters (see “LDAP_SEARCH” on page 131 for more information on filters).

If **idsldapsearch** 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 **idsldapsearch**, type `idsldapsearch -?`.

Note: The search filter size limit is set at 4 KB in the `ldapsearch.c` file. Any filter size larger than 4 KB will be rejected by the `idsldapsearch` utility. If you want to change `ldapsearch.c` to handle a filter larger than 4 KB (even though an altered version of `idsldapsearch` will not be supported), then change the following line in `ldapsearch.c`:

```
#define FILTERSIZE 4096
```

to something like the following:

```
#define FILTERSIZE 16000
```

You must recompile `ldapsearch.c` for these changes to take effect.

Options

-a deref

Specify how aliases dereferencing is done. `deref` should be one of `never`, `always`, `search`, or `find` to specify that aliases are never dereferenced, always dereferenced, dereferenced when searching, or dereferenced only when locating the base object for the search. The default is to never dereference aliases.

-A Retrieve attributes only (no values). This is useful when you 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 will examine the `LDAP_BASEDN` environment variable for a `searchbase` definition. If neither is set, the default base is set to `""`, which is a null search. A null search returns all the entries in the entire Directory Information Tree (DIT). This search requires a **-s** subtree option. Otherwise, an error message is displayed. Be aware that null based search requests consume a lot of resource.

-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 `idsldapsearch` 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, `idsldapsearch` 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 `idsldapsearch` 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 only contain UTF-8 (or base-64 encoded UTF-8) representations of string data. See "IANA character sets supported by platform" on page 206 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>

Sets the LDAP debugging level to `<debuglevel>`. This option causes the

utility to generate debug output to stdout. The *<debuglevel>* is a bit mask that controls which output is generated with values up to 65535. This parameter is for use by IBM service personnel. See "Debugging levels" in the *IBM Tivoli Directory Server Version 6.0 Administration Guide* for additional information on debug levels.

-D binddn

Use *binddn* to bind to the LDAP directory. *binddn* is a string-represented DN. When used with -m DIGEST-MD5, it specifies the authorization ID. It can be either a DN or an authzId string that starts with "u:" or "dn:".

-e Display the LDAP library version information and exits.

-f file Perform sequence of searches using filters in *file*. "%s" must be substituted for the filter.

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

-G realm

Specify the name of the realm. When used with the -m DIGEST-MD5, the value is passed to the server during the bind.

-h ldaphost

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

-i 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 "-" character, then the lines are read from standard input.

For example, in the command, `idsldapsearch -V3 -v -b "o=ibm,c=us" -D "cn=admin" -w ldap -i filter.input %s dn`, the *filter.input* file might contain the following filter information:

```
(cn=*Z)
(cn=*Z*)
(cn=Z*)
(cn=*Z*)
(cn~=A)
(cn>=A)
(cn<=B)
```

Note: Each filter must be specified on a separate line.

The command performs a search of the subtree **o=ibm,c=us** for each of the filters beginning with **cn=*Z**. When that search is completed, the search begins for the next filter **cn=*Z*** and so forth until the search for the last filter **cn<=B** is completed.

Note: The -i *<file>* option replaces the -f *<file>* option. The -f option is still supported, although it is deprecated.

-k Use server administration control on bind.

-K keyfile

Specify the name of the SSL or TLS 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 will first look 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 will be used, if present.

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

- AIX operating systems - /opt/IBM/ldap/V6.0
- HP-UX operating systems - /opt/IBM/ldap/V6.0
- Linux operating systems - /opt/ibm/ldap/V6.0
- Solaris operating systems - /opt/IBM/ldap/V6.0
- Windows operating systems - <local_drive>:\Program Files\IBM\LDAP\V6.0 (This is the default install location. The actual IDS_LDAP_HOME is determined during installation.)

See 154 for more information about default key database files, and default Certificate Authorities.

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 certificate authorities (CAs) that are trusted by the client. These types of X.509 certificates are also known as trusted roots. For more information on managing an SSL or TLS key database, see Chapter 4, "Using gsk7IKM," on page 171. Also see the "Security functions" on page 50 below and LDAP SSL or TLS APIs 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 will be 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 Show what would be done, but don't actually modify entries. Useful for debugging in conjunction with **-v**.

-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 *attr_type*

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, with the given name (givenname) 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 `idsldapsearch` operation is not to sort the returned results.

-O *maxhops*

Specify `maxhops` 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 may 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 *pagesize*

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 `idsldapsearch` 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, `idsldapsearch` 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 `idsldapsearch` operation is to return all entries in a single request. No paging is done for the default `idsldapsearch` operation.

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

-s scope

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

Note: If you specify a null search, either by not specifying a **-b** option or specifying **-b ""**, you must the **-s** option. The default scope is disabled for a null search.

-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.

-U username

Specifies the username. This is required with **-m DIGEST-MD5** and ignored when any other mechanism is used. The value *username* depends on what attribute the server is configured to use. It might be a `uid` or any other value that is used to locate the entry.

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

-V Specifies the LDAP version to be used by `idsldapmodify` 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 `idsldapmodify`, selects LDAP V3 as the preferred protocol by using `ldap_init` instead of `ldap_open`.

-w password | ?

Use *password* as the password for authentication. Use the `?` to generate a password prompt. Using this prompt prevents your password from being visible through the `ps` command.

-x Use FIPS mode processing (SSL/TLS only).

-y proxydn

Specifies the DN to be used for proxied authorization.

-Y Use a secure TLS connection to communicate with the LDAP server. The **-Y** option is only supported when IBM's GSKit, is installed.

-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 component entry, as provided by IBM's GSKit, is installed.

-9 p Sets criticality for paging to false. The search is handled without paging.

- 9 s Sets criticality for sorting to false. The search is handled without sorting.
- 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.

Output format

If one or more entries are found, each entry is written to standard output in the form:

```

Distinguished Name (DN)

attributename=value

attributename=value

attributename=value

...

```

Multiple entries are separated with a single blank line. If the -F option is used to specify a separator character, it will be used instead of the `=' 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:

```
idsldapsearch "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 is retrieved and printed to standard output. The output might look something like this if two entries are found:

```
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:

```
idsldapsearch -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/idsldapsearch-audio-a19924
```

```
jpegPhoto=/tmp/idsldapsearch-jpegPhoto-a19924
```

This command:

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

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

```
dn: o=University of Alaska Fairbanks, c=US
```

```
o: University of Alaska Fairbanks
```

```
description: Preparing Alaska for a brave new tomorrow
```

```

description: leaf node only

dn: o=University of Colorado at Boulder, c=US
o: University of Colorado at Boulder
description: No personnel information
description: Institution of education and research

dn: o=University of Colorado at Denver, c=US
o: University of Colorado at Denver
o: UCD
o: CU/Denver
o: CU-Denver
description: Institute for Higher Learning and Research

dn: o=University of Florida, c=US
o: University of Florida
o: UF1
description: Shaper of young minds

...

```

This command:

```
idsldapsearch -b "o=ibm,c=us" -o ibm-slapdDN "objectclass=person" ibm-slapdDN
```

performs a subtree level search at the o=ibm,c=us level for all persons. When this special attribute is used for sorted searches, the search results are sorted by the string representation of the Distinguished Name (DN). The output might look something like this:

```

cn=Al Edwards,ou=Widget Division,ou=Austin,o=IBM,c=US
cn=Al Garcia,ou=Home Entertainment,ou=Austin,o=IBM,c=US
cn=Amy Nguyen,ou=In Flight Systems,ou=Austin,o=IBM,c=US
cn=Arthur Edwards,ou=Widget Division,ou=Austin,o=IBM,c=US
cn=Becky Garcia,ou=In Flight Systems,ou=Austin,o=IBM,c=US
cn=Ben Catu,ou=In Flight Systems,ou=Austin,o=IBM,c=US
cn=Ben Garcia Jr,ou=Home Entertainment,ou=Austin,o=IBM,c=US
cn=Bill Keller Jr.,ou=In Flight Systems,ou=Austin,o=IBM,c=US
cn=Bob Campbell,ou=In Flight Systems,ou=Austin,o=IBM,c=US

```

This command:

```
idsldapsearch -b "o=ibm,c=us" -s base "objectclass=*" numSubordinates
```

performs a one-level search at the o=ibm,c=us level and returns the number of entries that would be returned by a one-level search. The count returned does not take into account whether the bound client has authority to read any of the entries that are included in the count, other than the entry containing this value. If you have loaded the example file **sample.ldif** and issued the specified command with the numSubordinates attribute, the result is:

```
o=IBM,c=US  
numSubordinates=2
```

Security functions

To use the SSL or TLS -related functions associated with this utility, see “SSL, TLS notes” on page 55.

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.

See also

idsldapadd, idsldapchangepwd, idsldapdelete, idsldapexop, idsldapmodify, idsldapmodrdn

idsldaptrace, ldaptrace

The administration tracing utility. This utility is to be used in conjunction with IBM support to solve specific problems.

Notes:

1. Only the administrator or a member of the administrative group can use this utility.
2. Using **idsldaptrace** consumes resources and affects the performance of the server.

Synopsis

```
idsldaptrace | ldaptrace [-a port -l [on|off|clr|chg|info|dump] --[ldtrc options]  
-d debugLevel -D adminDn -h hostname -K keyfile -m debugLevel  
-N key_name -o debugFile -p port -P key_pw -t [start|stop]  
-v -w adminPW[? -x -Z] -?
```

Description

The administration tracing utility, **idsldaptrace**, is used to dynamically activate or deactivate tracing of the Directory Server. This extended operation can also be used to set the message level and specify the name of the file to the output is written. If LDAP trace facility (ldtrc) options are requested, they must be preceded by --.

To display syntax help for **idsldaptrace**, type: `idsldaptrace -?`

Note: While the **idsldaptrace** utility can be used with SSL or TLS, only the simple bind mechanism is supported.

Options

- a** *port* Specifies an alternate TCP port where IBM Administration Daemon (idsdiradm), not the Directory Server, is listening. The default port is 3538. If not specified and **-Z** is specified, the default SSL port 3539 is used.
- d** *debugLevel*
Debug this program.
- D** *adminDn*
Bind DN.
- h** *ldaphost*
Specify an alternate host on which the Directory Server and the Administration Daemon are running.
- K** *keyfile*
Specify the name of the SSL or TLS 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 will first look 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 will be used, if present.

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

- AIX operating systems - `/opt/IBM/ldap/V6.0`
- HP-UX operating systems - `/opt/IBM/ldap/V6.0`
- Linux operating systems - `/opt/ibm/ldap/V6.0`
- Solaris operating systems - `/opt/IBM/ldap/V6.0`
- Windows operating systems - `<local_drive>:\Program Files\IBM\LDAP\V6.0` (This is the default install location. The actual `IDS_LDAP_HOME` is determined during installation.)

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 certificate authorities (CAs) that are trusted by the client. These types of X.509 certificates are also known as trusted roots. For more information on managing an SSL or TLS key database, see Chapter 4, "Using gsk7IKM," on page 171. Also see the "Security functions" on page 35 and "LDAP_SSL" on page 152 for more information about SSL and certificates.

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

-l [**on** | **off** | **clr** | **chg** | **info** | **dump**] **-[ldtrc options]**

- on** Turns on the tracing facility. You can specify any of the following `ldtrc` options preceded by an extra `-`.
 - `[-m <mask>]` where `<mask> = <products>.<events>.<components>.<classes>.<functions>`.
 - `[-p <pid>[.<tid>]]` traces only the specified process or thread.
 - `[-c <cpid>]` traces only the specified companion process.

- [-e <maxSeverErrors>] stops tracing after the maximum number of sever errors (maxSevereErrors) is reached.
- [-s | -f <fileName>] sends the output to shared memory or a file.
- [-l [<bufferSize>] | -i [<bufferSize>]] specifies to retain the last or the initial records. The default buffer is 1M.
- [-this <thisPointer>] trace only the specified object.

Note: The tracing facility must be on for server data to be traced.

off Turns off the tracing facility.

clr Clears the existing trace buffer.

chg The trace must be active before you can use the **chg** option to change the values for the following **ldtrc** options:

- [-m <mask>] where <mask> = <products>.<events>.<components>.<classes>.<functions>.
- [-p <pid>[.<tid>]] traces only the specified process or thread.
- [-c <cpid>] traces only the specified companion process.
- [-e <maxSeverErrors>] stops tracing after the maximum number of sever errors (maxSevereErrors) is reached.
- [-this <thisPointer>] trace only the specified object.

info Gets information about the trace. You must specify the source file which can be either a binary trace file, or trace buffer and a destination file. The following is an example of the information that the **info** parameter gives:

```
C:\>ldtrc info
Trace Version      :      1.00
Op. System        :      NT
Op. Sys. Version  :      4.0
H/W Platform      :      80x86

Mask              :      *.*.*.*.*
pid.tid to trace  :      all
cpid   to trace   :      all
this pointer to trace :      all
Treat this rc as sys err: none
Max severe errors :      1
Max record size   :      32768 bytes
Trace destination :      shared memory
Records to keep   :      last
Trace buffer size :      1048576 bytes
Trace data pointer check: no
```

dump Dumps the trace information to a file. This information includes process flow data as well as server debug messages. You can specify the name of the destination file where you want to dump the trace. The default destination files is:

For Unix-based systems:

`/var/ldap/ibmslapd.trace.dump.`

For Windows-based systems:

`<installationpath>\var\ibmslapd.trace.dump`

Note: This file contains binary ldtrc data that must be formatted with the **ldtrc format** command.

- m** *<debuglevel>*
Sets the LDAP debugging level to *<debuglevel>*. This option causes the utility to generate debug output to stdout. The *<debuglevel>* is a bit mask that controls which output is generated with values up to 65535. This parameter is for use by IBM service personnel. See "Debugging levels" in the *IBM Tivoli Directory Server Version 6.0 Administration Guide* for additional information on debug levels.
- 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 debugfile**
Specifies the output file name for the server debug messages.
- p port**
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 may 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.
- t [start | stop]**
 - start** Starts the collection of server trace data.
 - stop** Stops the collection of server trace data.
- v** Specifies to run in verbose mode.
- w** *adminPW* | ?
Use *adminPW* as the password for authentication. Use the ? to generate a password prompt. Using this prompt prevents your password from being visible through the **ps** command.
- x** Use FIPS mode processing (SSL/TLS only).
- Z** Use a secure LDAP connection (SSL).
- ?** Displays the syntax format.

Examples

To turn the ldtrc facility on and start the server trace with a 2M trace buffer, issue the command:

```
idsldaptrace -h <hostname> -D <adminDN> -w <adminpw> -l on -t start -- | 2000000
```

To stop the server trace, issue the command:

```
idsldaptrace -h <hostname> -D <adminDN> -w <adminpw> -t stop
```

To turn off the ldtrc facility, issue the command:
idsldaptrace -h <hostname> -D <adminDN> -w <adminpw> -l off

See also

"ldtrc" in the *IBM Tivoli Directory Server Version 6.0 Administration Guide* .

ibmdirctl, idssdirctl

The administration daemon control program. The administration daemon (**idsdiradm**) must be running. See "Starting an instance of the directory administration daemon" and "Directory administration daemon" in the *IBM Tivoli Directory Server Version 6.0 Administration Guide* .

Note: Only the administrator may use this utility.

Synopsis

```
ibmdirctl [-d | D adminDN] [-h | H hostname] [-K keyfile] [ -N key_name ]  
          [-p port] [-P key_pw] [-v] [-w | W adminPW | ?] [-Y] [-Z] [-?]  
command -- [idsslapd options]
```

where *command* is {start|stop|restart|status|statusreturn|admstop}

Description

The administration daemon control program, **ibmdirctl**, is used to start, stop, restart or query the status of the IBM Tivoli Directory Server. It can also be used to stop the administration daemon. If idsslapd options are requested, they must be preceded by the --.

To display syntax help for **ibmdirctl**, type `ibmdirctl -?`.

Options

- d | D adminDN**
Use adminDN to bind to the LDAP directory. The adminDN is a string-represented DN (see LDAP Distinguished Names).
- h | H hostname**
Specify an alternate host on which the ldap server and the admin daemon are running.
- K keyfile**
Specifies the file to use for keys.
- N key_name**
Specifies the private key name to use in keyfile.
- p port**
Specify an alternate TCP port where the admin daemon is listening. The default LDAP port is 3538.
- P key_pw**
Specifies the key file password.
- v**
Specifies to run in verbose mode.

-w | **W** *adminPW* | **?**

Use *adminPW* as the password for authentication. Use the **?** to generate a password prompt. Using this prompt prevents your password from being visible through the **ps** command.

-Y Specifies to use a secure LDAP connection (TLS).

-Z Specifies to use a secure LDAP connection (SSL).

-? Displays the syntax format.

command

- **start** - Start the server.
- **stop** - Stop the server.
- **restart** - Stop then start the server.
- **status** - Query the status the server.
- **statusreturn** - Sets exit code (0=running, 1=starting, 2=stopped)
- **admstop** - Stop the IBM Tivoli Directory Server administration daemon.

Note: The **stop** command can be issued directly to the ldap server.

-- idsslapd options

The only **idsslapd** options that the **idsslapd** process takes at startup time are:

- **-a** - Start the server in configuration only mode.
- **-n** - Do not start the server, if the server is unable to start with the database backends (no configuration only mode).

Notes:

1. If **idsslapd** options are requested, they must be preceded by the **--**.
2. The **idsslapd** options are ignored if the **stop**, **status** or **admstop** commands are issued.

Example

To start the server in configuration only mode issue the command:

```
ibmdirctl -h mymachine -D myDN -w mypassword -p 3538 start -- -a
```

To stop the server issue the command:

```
ibmdirctl -h mymachine -D myDN -w mypassword -p 3538 stop
```

SSL, TLS notes

To use the SSL or TLS -related functions associated with this utility, the SSL or TLS libraries and tools must be installed. The SSL or TLS libraries and tools are provided with IBM's Global Security Kit (GSKit), which includes security software developed by RSA Security Inc.

Note: For information regarding the use of 128-bit and triple DES encryption by LDAP applications, including the LDAP sample programs, see "LDAP_SSL" on page 152. This section describes the steps required to build the sample programs and your applications so they can use SSL with the strongest encryption algorithms available.

See the makefile associated with the sample programs for more information on linking an LDAP application so that it has access to 128-bit and triple-DES encryption algorithms.

The content of a client's key database file is managed with the `gsk7ikm` utility. For more information on this Java utility, see Chapter 4, "Using `gsk7IKM`," on page 171. The `gsk7ikm` 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 `gsk7ikm` 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 or TLS connection with the server are encrypted including the LDAP credentials that are supplied on the `ldap_bind` or `ldap_simple_bind_s`. For example, if the LDAP server is using a high-assurance VeriSign certificate, you should 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 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 or TLS connection with the server are encrypted, including the LDAP credentials that are supplied on the `ldap_bind` or `ldap_simple_bind_s`.
- Create a key pair using `gsk7ikm` 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.

Chapter 3. API categories

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

- “LDAP_ABANDON”
- “LDAP_ADD” on page 59
- “LDAP_BIND / UNBIND” on page 61
- “LDAP_CODEPAGE” on page 68
- “LDAP_COMPARE” on page 76
- “LDAP controls” on page 78
- “LDAP_CREATE_PROXYAUTH_CONTROL” on page 74
- “LDAP_DELETE” on page 80
- “LDAP_ERROR” on page 81
- “LDAP_EXTENDED_OPERATION” on page 88
- “LDAP_FIRST_ATTRIBUTE” on page 90
- “LDAP_FIRST_ENTRY, LDAP_FIRST_REFERENCE” on page 92
- “LDAP_GET_BIND_CONTROLS” on page 95
- “LDAP_GET_DN” on page 95
- “LDAP_GET_VALUES” on page 98
- “LDAP_INIT” on page 100
- “LDAP_MEMFREE” on page 111
- “LDAP_MESSAGE” on page 113
- “LDAP_MODIFY” on page 114
- “LDAP_PAGED_RESULTS” on page 117
- “LDAP_PARSE_RESULT” on page 120
- “LDAP_PASSWORD_POLICY” on page 123
- “LDAP_PLUGIN_REGISTRATION” on page 124
- “LDAP_RENAME” on page 127
- “LDAP_RESULT” on page 129
- “LDAP_SEARCH” on page 131
- “LDAP_SERVER_INFORMATION IN DNS” on page 135
- “LDAP_SSL” on page 152
- “LDAP_START_TLS” on page 159
- “LDAP_STOP_TLS” on page 161
- “LDAP_SSL_ENVIRONMENT_INIT” on page 164
- “LDAP_SORT” on page 165
- “LDAP_URL” on page 161

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 78 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 78 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()` API returns 0 if the abandon was successful or -1 if unsuccessful; it does not support LDAP V3 server controls or client controls. The `ldap_abandon_ext()` API returns the constant `LDAP_SUCCESS` if the abandon was successful, or another LDAP error code if not.

Errors

`ldap_abandon()` returns 0 if the operation is successful, -1 if unsuccessful, setting `ld_errno` appropriately. See “LDAP_ERROR” on page 81 for details.
`ldap_abandon_ext()` returns `LDAP_SUCCESS` if successful and returns an LDAP error code if unsuccessful.

See also

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 78 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 78 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. After 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 whether 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 81 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_modify

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.

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.

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`.
- `LDAP_MECHANISM_DIGEST_MD5` mechanism, represented by the string `DIGEST-MD5`.

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 the DN and credentials 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.

When **mechanism** is set 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()`.

See “LDAP_PLUGIN_REGISTRATION” on page 124 for more information about using LDAP client plug-ins. See Chapter 6, “LDAP client plug-in programming reference,” on page 185 for more information about developing an LDAP client plug-in.

The LDAP_MECHANISM_DIGEST_MD5 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 and prevents replay attacks.

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 DIGEST-MD5 mechanism fails. When using the DIGEST-MD5 mechanism, the hostname supplied on the `ldap_init` call must resolve to the fully qualified hostname of the server.

The application must supply a username on the `ldap_sasl_bind_s` call by using the IBM_CLIENT_MD5_USER_NAME_OID client control. The application can optionally supply a realm on the `ldap_sasl_bind_s` call by using the IBM_CLIENT_MD5_REALM_NAME_OID client control. The application can optionally supply an authorization ID as the `dn` parameter.

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.

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.

password

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 78 for more information about server controls.

clientctrls

Specifies a list of LDAP client controls. See “LDAP controls” on page 78 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 and the user's password (supplied in `password`). It returns an LDAP error indication (see "LDAP_ERROR" on page 81). 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.

The deprecated APIs 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

When the input parameter **mechanism** is set 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 Tivoli 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 124 for more information about using an LDAP client plug-in. See Chapter 6, “LDAP client plug-in programming reference,” on page 185 for more information about developing an LDAP client plug-in.

DIGEST-MD5 SASL binds: The server accepts SASL bind requests using the DIGEST-MD5 mechanism. There are two types of DIGEST-MD5 bind requests: Initial Authentication bind requests and Subsequent Authentication bind requests. Initial Authentication is required and supported by IBM Tivoli Directory Server. Subsequent Authentication support is optional, and is not supported by IBM Tivoli Directory Server.

The server responds to a DIGEST-MD5 SASL bind request with a digest-challenge. The challenge contains the values required by RFC2831 section 2.1.1, with the following implementation-specific behavior:

- **realm** - The server always sends the realm that the server is configured to be in.
- **nonce** - The server generates a random nonce.
- **qop-options** - The server supports “auth” only.

The next response from the client to the server must be another DIGEST-MD5 SASL bind message. The response includes several fields containing values that the server uses as follows:

- **username** - The server uses the username value to determine whether the user is binding as an administrator or to find an entry in the primary rdbm backend. If the username is an administrator’s DigestUsername, then the server uses that administrator to bind. If the username was not an administrator’s, then the server searches the primary rdbm for a user with that username. If the username doesn’t correspond to a single entry or the entry doesn’t have a userpassword value, the server returns LDAP_INVALID_CREDENTIALS. It will also print out the appropriate error message.
- **realm** - The value in the realm field must match the realm that the server is configured to be in. If the realm value does not match the realm that the server is configured to be in, the server returns LDAP_PROTOCOL_ERROR.
- **nonce** - The value in the nonce field must match the nonce value that the server sent the client with the digest-challenge. If the value does not match, the server returns LDAP_PROTOCOL_ERROR.
- **response** - The value in the response field contains a hash of the password. For each of the userpassword values that the server can get from the user entry, it generates the DIGEST-MD5 hash and compares it with the hash sent by the client. If one matches, the server returns LDAP_SUCCESS and the user is bound as that user. Otherwise, the server returns LDAP_INVALID_CREDENTIALS and prints out an error message.
- **authzid** - The value in the authzid field can contain a “dn:”- or “u:”-style authorization ID from RFC 2829 that the server will use for authority checking after the bind, rather than the entry found for the username, similar to Proxied Authentication. The entry that the username corresponds to needs to have the authority to use the other DN. If the authzid contains a “u:”-style authorization ID, the server maps the value to an entry the same as was done for the username parameter. If the mapping fails the server returns LDAP_INVALID_CREDENTIALS.

User-supplied SASL plug-ins: The application developer, or a third party, can implement additional SASL mechanisms using the IBM Tivoli Directory Server C-Client 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 `ibmldap.conf`, the application must explicitly register the plug-in, using the `ldap_register_plugin()` API. See "Defining a SASL plug-in" for information about defining a SASL plug-in for use with an application. See "LDAP_PLUGIN_REGISTRATION" on page 124 for more information about using an LDAP client plug-in. See Chapter 6, "LDAP client plug-in programming reference," on page 185 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 `supportedsaslm mechanisms` 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 `ibmldap.conf` file. By default, the IBM Tivoli Directory Server C-Client `cram-md5` plug-in is defined in `ibmldap.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 126 for more information about locating a plug-in library and defining plug-ins in the `ibmldap.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 because 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 available only 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` nonzero to give the application a chance to free any memory allocated in the previous call.

If anything other than `LDAP_SUCCESS` is returned by the first call to the `rebindproc`, referral processing is stopped and the 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_error`, `ldap_open`

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(
    const 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 206 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_FRFRIS08859_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 206 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 nonzero LDAP return code if an error occurs. See "LDAP_ERROR" on page 81 for more details.

See also

`ldap_error`

LDAP_CREATE_PROXYAUTH_CONTROL

`ldap_create_proxyauth_control`

`ldap_proxy_dn_prefix`

Purpose

Creates an LDAP control that will allow a bind entity to assume a proxy identity.

Synopsis

```
#include <ldap.h>

int ldap_create_proxyauth_control(
    LDAP *ld,
    char *proxyDN,
    int iscritical,
    LDAPControl **controlp)

int ldap_proxy_dn_prefix(
    char **proxyDN,
    char *param)
```

Input parameters

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

proxyDN Specifies the DN of the entry whose identity the client will assume.

iscritical Specifies whether the persistent search control is critical to the current operation. This should be set to a non-zero value.

controlp Pointer to a pointer of a structure that is created by this function. This control should be freed by calling `ldap_control_free()` function, when it is done using the control.

Usage

This API is used to create an LDAP control containing the proxy authorization identity. The created proxy authorization control will then be included in LDAP operations to request an operation from the server.

Using the proxy authorization control mechanism, a client can bind to the LDAP directory using its own identity, but is granted proxy authorization rights of another user to access the target directory.

When the LDAP server receives an operation with proxy authorization control, the bind DN is validated against the administrative group and/or the predefined proxy authorization group to determine whether the bind DN should be granted the proxy authorization right. In other words, the bound application client must be a member of the administrative group or proxy authorization group in order to request a proxy authorization operation.

For a specific DN, the `ldap_proxy_dn_prefix` function ensures that the DN has the proxy DN prefix. The DN is passed in using the `param` parameter. The value is returned using the `proxyDN` parameter. If the passed in DN already has the "dn:" prefix, the parameter is simply copied into the return value. If the passed in DN does not have the "dn:" prefix, then a new string is allocated with the "dn:" prefix. The return code can be:

- `LDAP_PARAM_ERROR` if the `param` is null
- `LDAP_NO_MEMORY` if the function failed to allocate memory
- `LDAP_SUCCESS` if a new proxyDN was successfully allocated

If LDAP_SUCCESS is returned, it is the caller's responsibility to free the returned proxyDN.

Errors

LDAP_PARAM_ERROR returns if an invalid parameter was passed.

LDAP_NO_MEMORY returns if memory cannot be allocated.

LDAP_ENCODING_ERROR returns if an error occurred when encoding the control.

LDAP_UNAVAILABLE_CRITICAL_EXTENSION returns if server does not support proxy authorization and iscritical is set to a non-zero value.

See also

ldap_controls, ldap_bind, ldap_search, ldap_modify, ldap_delete, ldap_add

LDAP_COMPARE

ldap_compare
ldap_compare_s
ldap_compare_ext
ldap_compare_ext_s

Purpose

Performs 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 on which to perform the comparison.

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 98

serverctrls

Specifies a list of LDAP server controls. This parameter can be set to NULL. See “LDAP controls” on page 78 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 78 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 it was not successfully sent. 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()` 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. The session error parameters can be obtained by using `ldap_get_errno()`.

See “LDAP_ERROR” on page 81 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 can be `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()` API returns an LDAP error code that can be interpreted by calling one of the `ldap_error` routines. The `ldap_compare()` API returns -1 if the initiation request was unsuccessful. It returns the message ID of the request if successful.

See also

`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 nonzero, the operation is carried out only if it is recognized and supported by the server or the client for client-side controls.

Functions to manipulate controls

`ldap_insert_control`
`ldap_add_control`
`ldap_remove_control`
`ldap_copy_controls`

Purpose

Add, remove, or copy controls.

Synopsis

```
#include <ldap.h>

int ldap_insert_control(
    LDAPControl *newControl,
    LDAPControl ***ctrlList);

int ldap_add_control(
    const char *oid, ber_len_t len ,
    char *value,
    int isCritical,
    LDAPControl ***ctrlList);

int ldap_remove_control(
    LDAPControl *delControl,
    LDAPControl ***ctrlList,
    int freeit);

int ldap_copy_controls(
    LDAPControl ***to_here,
    LDAPControl **from);
```

Input parameters

newcontrol

Specifies a control to be inserted into a list of controls.

ctrlList

Specifies a list of LDAP server controls

oid Specifies the control type, represented as a string.

len Specifies the length of the value string.

value Specifies the data associated with the control.

isCritical

Specifies whether the control is critical or not.

delControl

Specifies the control to be deleted.

freeit Specifies whether or not to free the control. If set to TRUE, the control will be freed. If set to FALSE, the control will not be freed.

to_here

Specifies the location to which to copy the control list.

from

Specifies the location of the control list to be copied.

Usage

The `ldap_insert_control()` API inserts the control **newcontrol* into a list of controls specified by ****ctrlList*. The function will allocate space in the list for the control, but will not allocate the actual control. Returns LDAP_SUCCESS if the request was successfully sent or LDAP_NO_MEMORY if the control could not be inserted.

The `ldap_add_control()` API creates a control (using the *oid*, *len*, *value* and *isCritical* values) and inserts it into a list of controls specified by ****ctrlList*. The function will allocate space in the list for the control. Returns LDAP_SUCCESS if the request was successfully sent or LDAP_NO_MEMORY if the control could not be added.

The `ldap_remove_control()` API removes the control from the list. If *freeit* is not 0, the control will be freed. If *freeit* is set to 0, the control will not be freed. Returns `LDAP_SUCCESS` if the request was successfully sent or `LDAP_NO_MEMORY` if the control could not be removed.

The `ldap_copy_controls()` API makes a copy of the control list. Returns `LDAP_SUCCESS` if the request was successfully sent or `LDAP_NO_MEMORY` if the control list could not be copied.

LDAP_DELETE

```
ldap_delete
ldap_delete_s
ldap_delete_ext
ldap_delete_ext_s
```

Purpose

Performs 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 78 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 78 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 returns 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. The `ldap_parse_result()` API 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 an 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” 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 returns 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 that can be interpreted by calling an `ldap_error` routine. The `ldap_delete()` API returns -1 if the request initiation was unsuccessful. It returns the message ID of the request if successful.

See also

`ldap_error`

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)

const char *ldap_err2string(
    int err)

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** Specifies 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 nonzero, the result, **res**, is freed by the call. If zero, **res** is not freed by the call.

errnum

Specifies 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"). If the `freeit` parameter is nonzero, 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 shown in the following tables.

Table 1. General return codes

Dec value	Value	Hex value	Brief description	Detailed description
00	LDAP_SUCCESS	00	Success	The request was successful.
00	LDAP_OPERATIONS_ERROR	01	Operations error	An operations error occurred.

Table 1. General return codes (continued)

Dec value	Value	Hex value	Brief description	Detailed description
02	LDAP_PROTOCOL_ERROR	02	Protocol error	A protocol violation was detected.
03	LDAP_TIMELIMIT_EXCEEDED	03	Time limit exceeded	An LDAP time limit was exceeded.
04	LDAP_SIZELIMIT_EXCEEDED	04	Size limit exceeded	An LDAP size limit was exceeded.
05	LDAP_COMPARE_FALSE	05	Compare false	A compare operation returned false.
06	LDAP_COMPARE_TRUE	06	Compare true	A compare operation returned true.
07	LDAP_STRONG_AUTH_NOT_SUPPORTED	07	Strong authentication not supported	The LDAP server does not support strong authentication.
08	LDAP_STRONG_AUTH_REQUIRED	08	Strong authentication required	Strong authentication is required for the operation.
09	LDAP_PARTIAL_RESULTS	09	Partial results and referral received	Partial results only returned.
10	LDAP_REFERRAL	0A	Referral returned	Referral returned.
11	LDAP_ADMIN_LIMIT_EXCEEDED	0B	Administration limit exceeded	Administration limit exceeded.
12	LDAP_UNAVAILABLE_CRITICAL_EXTENSION	0C	Critical extension not supported	Critical extension is not supported.
13	LDAP_CONFIDENTIALITY_REQUIRED	0D	Confidentiality is required	Confidentiality is required.
14	LDAP_SASLBIND_IN_PROGRESS	0E	SASL bind in progress	An SASL bind is in progress.
16	LDAP_NO_SUCH_ATTRIBUTE	10	No such attribute	The attribute type specified does not exist in the entry.
17	LDAP_UNDEFINED_TYPE	11	Undefined attribute type	The attribute type specified is not valid.
18	LDAP_INAPPROPRIATE_MATCHING	12	Inappropriate matching	Filter type not supported for the specified attribute.
19	LDAP_CONSTRAINT_VIOLATION	13	Constraint violation	An attribute value specified violates some constraint (for example, a postal address has too many lines, or a line that is too long).
20	LDAP_TYPE_OR_VALUE_EXISTS	14	Type or value exists	An attribute type or attribute value specified already exists in the entry.
21	LDAP_INVALID_SYNTAX	15	Invalid syntax	An attribute value that is not valid was specified.

Table 1. General return codes (continued)

Dec value	Value	Hex value	Brief description	Detailed description
32	LDAP_NO_SUCH_OBJECT	20	No such object	The specified object does not exist in the directory.
33	LDAP_ALIAS_PROBLEM	21	Alias problem	An alias in the directory points to a nonexistent entry.
34	LDAP_INVALID_DN_SYNTAX	22	Invalid DN syntax	A DN that is syntactically not valid was specified.
35	LDAP_IS_LEAF	23	Object is a leaf	The object specified is a leaf.
36	LDAP_ALIAS_DEREF_PROBLEM	24	Alias dereferencing problem	A problem was encountered when dereferencing an alias.
48	LDAP_INAPPROPRIATE_AUTH	30	Inappropriate authentication	Inappropriate authentication was specified (for example, LDAP_AUTH_SIMPLE was specified and the entry does not have a userPassword attribute).
49	LDAP_INVALID_CREDENTIALS	31	Invalid credentials	Invalid credentials were presented (for example, the wrong password).
50	LDAP_INSUFFICIENT_ACCESS	32	Insufficient access	The user has insufficient access to perform the operation.
51	LDAP_BUSY	33	DSA is busy	The DSA is busy.
52	LDAP_UNAVAILABLE	34	DSA is unavailable	The DSA is unavailable.
53	LDAP_UNWILLING_TO_PERFORM	35	DSA cannot perform	The DSA cannot perform the operation.
54	LDAP_LOOP_DETECT	36	Loop detected	A loop was detected.
64	LDAP_NAMING_VIOLATION	40	Naming violation	A naming violation occurred.
65	LDAP_OBJECT_CLASS_VIOLATION	41	Object class violation	An object class violation occurred (for example, a "required" attribute was missing from the entry).
66	LDAP_NOT_ALLOWED_ON_NONLEAF	42	Operation not allowed on nonleaf	The operation is not allowed on a nonleaf object.
67	LDAP_NOT_ALLOWED_ON_RDN	43	Operation not allowed on RDN	The operation is not allowed on an RDN.
68	LDAP_ALREADY_EXISTS	44	Already exists	The entry already exists.
69	LDAP_NO_OBJECT_CLASS_MODS	45	Cannot modify object class	Object class modifications are not allowed.
70	LDAP_RESULTS_TOO_LARGE	46	Results too large	Results too large.
71	LDAP_AFFECTS_MULTIPLE_DSAS	47	Affects multiple DSAs	Affects multiple DSAs.

Table 1. General return codes (continued)

Dec value	Value	Hex value	Brief description	Detailed description
80	LDAP_OTHER	50	Unknown error	An unknown error occurred.
81	LDAP_SERVER_DOWN	51	Can't contact LDAP server	The LDAP library cannot contact the LDAP server.
82	LDAP_LOCAL_ERROR	52	Local error	Some local error occurred. This is usually a failed memory allocation.
83	LDAP_ENCODING_ERROR	53	Encoding error	An error was encountered encoding parameters to send to the LDAP server.
84	LDAP_DECODING_ERROR	54	Decoding error	An error was encountered decoding a result from the LDAP server.
85	LDAP_TIMEOUT	55	Timed out	A time limit was exceeded while waiting for a result.
86	LDAP_AUTH_UNKNOWN	56	Unknown authentication method	The authentication method specified on a bind operation is not known.
87	LDAP_FILTER_ERROR	57	Bad search filter	An invalid filter was supplied to ldap_search (for example, unbalanced parentheses).
88	LDAP_USER_CANCELLED	58	User cancelled operation	The user cancelled the operation.
89	LDAP_PARAM_ERROR	59	Bad parameter to an LDAP routine	An LDAP routine was called with a bad parameter (for example, a NULL ld pointer, etc.).
90	LDAP_NO_MEMORY	5A	Out of memory	A memory allocation (for example malloc) call failed in an LDAP library routine.
91	LDAP_CONNECT_ERROR	5B	Connection error	Connection error.
92	LDAP_NOT_SUPPORTED	5C	Not supported	Not supported.
93	LDAP_CONTROL_NOT_FOUND	5D	Control not found	Control not found.
94	LDAP_NO_RESULTS_RETURNED	5E	No results returned	No results returned.
95	LDAP_MORE_RESULTS_TO_RETURN	5F	More results to return	More results to return.
96	LDAP_URL_ERR_NOTLDAP	60	URL doesn't begin with ldap://	The URL does not begin with ldap://.
97	LDAP_URL_ERR_NODN	61	URL has no DN (required)	The URL does not have a DN (required).
98	LDAP_URL_ERR_BADSCOPE	62	URL scope string is invalid	The URL scope string is not valid.

Table 1. General return codes (continued)

Dec value	Value	Hex value	Brief description	Detailed description
99	LDAP_URL_ERR_MEM	63	Can't allocate memory space	Cannot allocate memory space.
100	LDAP_CLIENT_LOOP	64	Client loop	Client loop.
101	LDAP_REFERRAL_LIMIT_EXCEEDED	65	Referral limit exceeded	Referral limit exceeded.
112	LDAP_SSL_ALREADY_INITIALIZED	70	ldap_ssl_client_init successfully called previously in this process	The ldap_ssl_client_init was successfully called previously in this process.
113	LDAP_SSL_INITIALIZE_FAILED	71	Initialization call failed	SSL Initialization call failed.
114	LDAP_SSL_CLIENT_INIT_NOT_CALLED	72	Must call ldap_ssl_client_init before attempting to use SSL connection	Must call ldap_ssl_client_init before attempting to use the SSL connection.
115	LDAP_SSL_PARAM_ERROR	73	Invalid SSL parameter previously specified	An SSL parameter that was not valid was previously specified.
116	LDAP_SSL_HANDSHAKE_FAILED	74	Failed to connect to SSL server	Failed to connect to SSL server.
117	LDAP_SSL_GET_CIPHER_FAILED	75	Not used	Deprecated
118	LDAP_SSL_NOT_AVAILABLE	76	SSL library cannot be located	Ensure that GSKit has been installed
128	LDAP_NO_EXPLICIT_OWNER	80	No explicit owner found	No explicit owner was found
129	LDAP_NO_LOCK	81	Could not obtain lock	Client library was not able to lock a required resource

In addition, the following DNS-related error codes are defined in the ldap.h file:

Table 2. DNS-related return codes

Dec value	Value	Hex value	Detailed description
133	LDAP_DNS_NO_SERVERS	85	No LDAP servers found
134	LDAP_DNS_TRUNCATED	86	Warning: truncated DNS results
135	LDAP_DNS_INVALID_DATA	87	Invalid DNS Data
136	LDAP_DNS_RESOLVE_ERROR	88	Can't resolve system domain or nameserver
137	LDAP_DNS_CONF_FILE_ERROR	89	DNS Configuration file error

The following UTF8-related error codes are defined in the ldap.h file:

Table 3. UTF8-related return codes

Dec value	Value	Hex value	Detailed description
160	LDAP_XLATE_E2BIG	A0	Output buffer overflow

Table 3. UTF8-related return codes (continued)

Dec value	Value	Hex value	Detailed description
161	LDAP_XLATE_EINVAL	A1	Input buffer truncated
162	LDAP_XLATE_EILSEQ	A2	Unusable input character
163	LDAP_XLATE_NO_ENTRY	A3	No codeset point to map to
176	LDAP_REG_FILE_NOT_FOUND	B0	NT Registry file not found
177	LDAP_REG_CANNOT_OPEN	B1	NT Registry cannot open
178	LDAP_REG_ENTRY_NOT_FOUND	B2	NT Registry entry not found
192	LDAP_CONF_FILE_NOT_OPENED	C0	Plug-in configuration file not opened
193	LDAP_PLUGIN_NOT_LOADED	C1	Plug-in library not loaded
194	LDAP_PLUGIN_FUNCTION_NOT_RESOLVED	C2	Plug-in function not resolved
195	LDAP_PLUGIN_NOT_INITIALIZED	C3	Plug-in library not initialized
196	LDAP_PLUGIN_COULD_NOT_BIND	C4	Plug-in function could not bind
208	LDAP_SASL_GSS_NO_SEC_CONTEXT	D0	gss_init_sec_context failed

See also

ldap_memfree, ldap_parse routines

LDAP_EXTENDED_OPERATION

ldap_extended_operation
 ldap_extended_operation_s

Purpose

Performs 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 78 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 78 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 `BerValue` structure pointer that is set to an allocated copy of the data returned by the server. This `BerValue` 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 is returned if the operation was not successful. 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 it returns another LDAP error

code if it was not successful. 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 Tivoli Directory Server Version 6.0 provides a server plug-in interface that can be used to add extended operation support. For more information, see the *IBM Tivoli Directory Server Version 6.0: 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.

A list of OIDs for supported extended operations can be found in Appendix F, “Object Identifiers (OIDs) for extended operations and controls,” on page 211.

Errors

The `ldap_extended_operation_s()` API returns the LDAP error code for the operation.

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

See “LDAP_ERROR” on page 81 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_result`, `ldap_error`

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()`.
- entry** Pointer to the `LDAPMessage` representing an entry.

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` that 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()`, -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 and sets the error code to `LDAP_SUCCESS`. If an error occurs, NULL is returned and an error code is set. The memory allocated for the `BerElement` buffer must be freed using `ldap_ber_free()`.

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()` and `ldap_next_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.

The `ldap_get_errno()` API can be used to obtain the error code. See "LDAP_ERROR" on page 81 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_first_entry`, `ldap_get_values`, `ldap_memfree`, `ldap_error`

LDAP_FIRST_ENTRY, LDAP_FIRST_REFERENCE

`ldap_first_entry`
`ldap_next_entry`
`ldap_count_entries`
`ldap_get_entry_controls_np`
`ldap_first_reference`
`ldap_next_reference`
`ldap_count_references`
`ldap_parse_reference_np`

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()`. Supply NULL for this parameter to ignore the referrals field.

freeit Specifies a Boolean value that determines if the LDAP result chain, as specified by `ref`, is to be freed. Any nonzero value 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 list 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. The `ldap_next_entry()` API 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. The

ldap_get_entry_controls_np() and ldap_parse_reference_np() APIs 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” on page 81 for a description of possible error codes.

See also

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

LDAP_GET_BIND_CONTROLS

ldap_get_bind_controls

Purpose

Allows client using ldap_sasl_bind_s methods to get controls sent by the server.

Synopsis

```
int ldap_get_bind_controls LDAP_P(
    LDAP *ld,
    LDAPControl ***bind_controls );
```

Input parameters

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

bind_controls
Cannot be NULL.

Output parameters

bind_controls will have a copy of the bind controls, or NULL if there are no controls.

Usage

After calling ldap_sasl_bind_s, the application calls ldap_get_bind_controls to get a NULL-terminated array of controls that the server returned on the bind. The caller is responsible for freeing the controls by using ldap_controls_free(). If the caller hasn't called ldap_sasl_bind_s for the supplied ld, the client will set bind_controls to NULLreturn

Errors

LDAP_PARAM_ERROR: If bind_controls=NULL, error code if ld not valid

See also

ldap_copy_controls

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 simple form (as returned from `ldap_dn2ufn()`).
- rdn** Specifies the RDN to be exploded (as returned from `ldap_explode_dn()`).
- entry** Specifies the entry whose dn is to be retrieved.
- notypes** Specifies if type names are to be returned for each RDN. If nonzero, 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 simple 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 simple form as "John Doe, Widget Division, Austin, IBM, US". Space for the simple 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. The `ldap_explode_dn()` API 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, and not their types, be returned. 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. The ldap_explode_rdn() API 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().

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 handle compound RDNs. 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

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 81 for a description of possible error codes.

Notes

These routines allocate memory that the caller must deallocate.

See also

`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 that represent the attribute's values. The `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 81 for a description of possible error codes.

See also

`ldap_first_entry`, `ldap_first_attribute`, `ldap_error`

LDAP_INIT

`ldap_init`
`ldap_open` (deprecated)
`ldap_set_option`
`ldap_get_option`
`ldap_version`

Purpose

Initializes the LDAP library, opens a connection to an LDAP server, and gets or sets 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 Tivoli 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 are equivalent:

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

and

```
ld=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 example 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 examples are equivalent:

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

and

```
ld=ldap_init ("myserver", LDAP_PORT);
```

See "Locating default LDAP servers" on page 110 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 Tivoli 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 103 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 103 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 109 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

The `ldap_init()` API 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 that is used to identify the connection and maintain per-connection information.

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

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 102).

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()` APIs:

LDAP_OPT_SIZELIMIT

Get or set maximum number of entries that can be returned on a search operation.

LDAP_OPT_TIMELIMIT

Get or set maximum number of seconds to wait for search results.

LDAP_OPT_REFHOPLIMIT

Get or set maximum number of referrals in a sequence that the client can follow.

LDAP_OPT_DEREF

Get or set rules for following aliases at the server.

LDAP_OPT_REFERRALS

Get or set whether or not referrals must be followed by the client.

LDAP_OPT_DEBUG

Get or set debug options.

LDAP_OPT_SSL_CIPHER

Get or set SSL ciphers to use.

LDAP_OPT_SSL_TIMEOUT

Get or set SSL timeout for refreshing session keys.

LDAP_OPT_REBIND_FN

Get or set address of application's setrebindproc procedure.

LDAP_OPT_PROTOCOL_VERSION

Get or set LDAP protocol version to use (V2 or V3).

LDAP_OPT_SERVER_CONTROLS

Get or set default server controls.

LDAP_OPT_CLIENT_CONTROLS

Get or set default client library controls.

LDAP_OPT_UTF8_IO

Get or 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.

See "LDAP_SET_OPTION syntax for LDAP V2 applications" on page 109 for important information 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.

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. Therefore, 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. Therefore, 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 bitmap 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. The default value is "05040A090306".

Note: If you try to get an SSL cipher and you are not running on an SSL version of IBM Tivoli Directory Server, an error is returned.

Supported ciphers:

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

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 number of seconds specified, 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)
```

Note: If you use `LDAP_OPT_SSL_TIMEOUT` and you are not running on an SSL version of IBM Tivoli Directory Server, an error is returned.

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. Because 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_I0, (void*)LDAP_UTF8_XLATE_ON);
ldap_get_option( ld, LDAP_OPT_UTF8_I0, &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 that 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 */
    const 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.

ldapapi_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.

ldapapi_protocol_version

The highest LDAP protocol version supported by the implementation. For example, if LDAP V3 is the highest version supported then this field is set to 3.

ldapapi_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 nonzero 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 Tivoli 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 following example 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 );
}
```

An LDAP application is typically running as LDAP V2 when it uses `ldap_open()` to create the LDAP connection. An 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 in which 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, for example, `ibm.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 model avoids the possibility of conflicts that 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, because 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 application's 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_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

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

The ldap_ber_free() API 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.

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.

See also

ldap_controls

LDAP_MESSAGE

ldap_first_message
ldap_next_message
ldap_count_messages

Purpose

Steps 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 any of the three types of messages.

The ldap_first_message() and ldap_next_message() APIs return NULL when no more messages exist in the result set to be returned. NULL is also returned if an

error occurs while stepping through the entries. When such an error occurs, `ldap_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 81 for a description of possible error codes.

See also

`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

Performs 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              *freemods)

```

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 C, “LDAP distinguished names,” on page 201 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 78 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 78 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 type 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 the 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`.

The `mod_values` array is NULL-terminated. Because 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 it returns another LDAP error code if it is not successful. 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. When 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. 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 an 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 81 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.

The `ldap_modify_s()` API 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 nonzero, 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 81 for more details.

See also

`ldap_error`, `ldap_add`

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          *ld,
    unsigned long  pageSize,
    struct berval *cookie,
    const char    isCritical,
    LDAPControl   **control)

int ldap_parse_page_control(
    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. After 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 78 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.

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.

The `ldap_create_page_control()` API takes as input a page size and a cookie, and outputs an `LDAPControl` structure that 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. After 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. Use `ldap_controls_free()` to free the

memory used by the client application to hold the server controls when you are finished 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 must 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.

Errors

The sort routines return an LDAP error code if they encounter an error parsing the result. See "LDAP_ERROR" on page 81 for a list of the LDAP error codes.

Notes

Controls, `serverControls`, and `cookie` must be freed by the caller.

See also

`ldap_search`, `ldap_parse_result`

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          **matcheddn,
    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.

matcheddn

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 `matcheddn` 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 that 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 `LDAPMessage`. 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 nonzero 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 the `ldap_parse_sasl_bind_result()` and the `ldap_parse_extended_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 the result was 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 81 for a list of the LDAP error codes.

See also

ldap_error, ldap_result

LDAP_PASSWORD_POLICY

ldap_parse_pwdpolicy_reponse

ldap_pwdpolicy_err2string

Purpose

LDAP routines for extracting information from results returned in the Password Policy Control Structure.

Synopsis

```
#include <ldap.h>

int ldap_parse_pwdpolicy_response(LDAPCONTROL **serverControls,
    int *controlerr,
    int *controlwarn,
    int *controlres)

const char *ldap_pwdpolicy_err2string(int err);
```

Input parameters

serverControls

Specifies an array of LDAPCONTROL pointers returned by a previous call to ldap_parse_result().

controlerr

Specifies a pointer to the result parameter that is filled in with the LDAP Password Policy error code, which can be used as input to ldap_pwdpolicy_err2string(), so that a text description of the error can be obtained.

controlwarn

Specifies a pointer to the result parameter that is filled in with the LDAP Password Policy warning code, which can be used as input to ldap_pwdpolicy_err2string(), so that a text description of the warning can be obtained.

controlres

Specifies a pointer to the result parameter that is filled in with the LDAP Password Policy warning result value.

err Specifies an integer value returned from ldap_parse_pwdpolicy_response() containing the Password Policy warning or error code.

Usage

The ldap_parse_pwdpolicy_response() API is used to:

- Obtain the LDAP Password Policy error or warning codes from the Password Policy Response Control associated with an LDAPMessage message.
- Obtain the LDAP Password Policy warning result code from the Password Policy Response Control that is associated with the returned Password Policy warning code.
- This function takes in an array of LDAPCONTROL structure pointers, parses these structures and then returns three integers containing the Password Policy response values.

The `ldap_pwdpolicy_err2string()` API is used to convert the numeric LDAP Password Policy error or warning code, as returned by `ldap_parse_pwdpolicy_response()`, into a NULL-terminated character string that describes the error or warning. The character string is returned as static data and must not be freed by the application.

Errors

The `ldap_parse_pwdpolicy_response` routine returns an LDAP error code if it encounters an error parsing the result.

See “LDAP_ERROR” on page 81 for a list of the LDAP error codes.

See also

`ldap_parse_result`

LDAP_PLUGIN_REGISTRATION

`ldap_register_plugin`
`ldap_query_plugin`
`ldap_free_query_plugin`

Purpose

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 `ibmldap.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 the plug-in type. The only type currently supported is sasl.

subtype

NULL-terminated string that specifies the subtype of the 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 only the file name with or without the file extension. If only 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);

```

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 is allocated.

Usage

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

- The plug-in is defined in the ibmldap.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 the `ibmldap.conf` file 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 `ibmldap.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 `ibmldap.conf` file. Note that the attempt to locate the `ibmldap.conf` file 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 `ibmldap.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, the application can use the `ldap_register_plugin()` API to add additional plug-in definitions, or to override definitions obtained from the `ibmldap.conf` file.

The `ibmldap.conf` file: The `ibmldap.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 `ibmldap.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 `ibmldap.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 idsldap_plugin_sasl_cram-md5 ldap_plugin_init
plugin sasl fpauth x:\security\fp1ib 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, standard operating system search rules are applied.

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

The algorithm used to locate the `ibmldap.conf` file is platform specific:

- On a UNIX system, the following search order is used:
 1. Query the environment variable `IBMLDAP_CONF` for the path to the `ibmldap.conf` file.
 2. Look for the `ibmldap.conf` file 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 the `ibmldap.conf` file.
 2. Look in the current directory for the `ibmldap.conf` file.
 3. Look for the `ibmldap.conf` file in the `\etc` directory under the LDAP installation directory; for example, `c:\Program Files\IBM\LDAP\v6.0\etc`.

If the definition for a SASL plug-in is not 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_error`

LDAP_RENAME

`ldap_rename`
`ldap_rename_s`
`ldap_modrdn`
`ldap_modrdn_s`

Purpose

Perform an LDAP rename operation.

Synopsis

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

```
int ldap_rename(  
    LDAP  
    const char *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 `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.
- 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 has meaning only 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 78 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 78 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.

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 whether 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.

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 81 for more details.

See also

`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 has meaning only 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 an LDAP parsing routine such as `ldap_first_entry()`.

If `ldap_result()` is unsuccessful, it returns -1 and sets the appropriate LDAP error, which can be retrieved by using `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)
#define LDAP_RES_RENAME          LDAP_RES_MODRDN

```

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_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. The attrsonly parameter must be set to 1 to request attribute types only or 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 78 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 78 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 129).

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 an LDAP error code if the operation was not successful. See “LDAP_ERROR” on page 81 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 and 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=*)"`. The `attrs` parameter optionally contains the list of attributes to return.

Listing the children of an entry

LDAP does not support a list operation directly. Instead, this operation is emulated by a search with base set to the DN of the list entry, scope set to `LDAP_SCOPE_ONELEVEL`, and filter set to `"(objectclass=*)"`. The `attrs` parameter optionally contains the list of attributes to return for each child entry. If only the distinguished names of child entries are desired, the `attrs` parameter must specify a NULL-terminated array of one-character strings that 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. The session error can be obtained by using `ldap_get_errno()`.

See "LDAP_ERROR" on page 81 for more details.

Notes

These routines allocate storage returned by the `res` parameter. Use `ldap_msgfree()` to free this storage.

See also

`ldap_result`, `ldap_error`, `ldap_sort`, `ldap_paged_results`

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. `search_source` can be one of the following:

- 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.
- Search the local LDAP DNS configuration file only.
- Search DNS only.

conf_filename

Specifies an alternative configuration filename. Specify NULL to get the default filename and location.

reserved

Represents a reserved area for future function, which must be initialized to zero.

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 does not 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 142 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 nonzero 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` will become the preferred protocol string for publishing LDAP and other services in DNS. If the application does not 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 file name. Specify `NULL` to get the default file name 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, the `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 the `ldap.h` file 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 does not 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.

- After 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 frees 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 nonzero 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 in the following platform-specific location:

UNIX /etc/ldap_server_info.conf

Windows NT and Windows 2000
\\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 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 nonzero 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: IBMNetwork"
             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      "ldatype: master"

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

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

ldapr1              A        199.23.45.297
```

```

                                TXT    "service:ldap://ldapr1:389/o=foo,c=us"
                                TXT    "ldatype: replica"

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

ldapr2sec                       A      199.23.45.298
                                TXT    "service:ldaps://ldapr2/o=foo,c=us"
                                TXT    "ldatype: 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    "ldatype: master"

```

If TXT records are not associated with the A record, defaults are assumed for port and `ldatype`.

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 81 for more details.

See also

`ldap_error`

LDAP_SSL

```
ldap_ssl_client_init
ldap_ssl_init
ldap_ssl_start (deprecated)
ldap_set_cipher
ldap_ssl_set_fips_mode_np
```

Purpose

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

For `ldap_ssl_set_fips_mode_np()`, the FIPS processing mode is set prior to creating an SSL environment used for securing server connections.

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,
    const char *name)

int ldap_ssl_start(
    LDAP *ld,
```

```

char    *keyring,
char    *keyring_pw,
char    *name)

int ldap_set_cipher(
LDAP    *ld,
char    *option)

int ldap_ssl_set_fips_mode_np(
int     mode)

```

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 Tivoli 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 110 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 `/etc` directory under `<LDAPHOME>`, where `<LDAPHOME>` is the path to the installed LDAP support. `<LDAPHOME>` varies by operating system platform:

- AIX - `/usr/ldap`
- Solaris - `/usr/IBMLdaps`
- HP-UX - `/usr/IBMLdap`
- Windows - `C:\Program Files\IBM\LDAP`

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

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>>\etc`, 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 `gsk7ikm` 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 `gsk7ikm` 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 `gsk7ikm` utility.

For more information on using `gsk7ikm` to manage the contents of a key database, see Chapter 4, "Using `gsk7IKM`," on page 171.

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 `gsk7ikm` 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 `gsk7ikm` 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 `gsk7IKM`," on page 171. 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.

mode For `ldap_ssl_set_fips_mode_np()`, **mode** specifies whether FIPS processing mode should be on (1) or off (0).

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 that uses the IBM Tivoli Directory Server C-Client SDK Version 6.0 with the required level of GSKit 6.0.3 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 100 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:

```
AES_256
AES_128
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 100 for more information about `ldap_set_option`.

`ldap_ssl_set_fips_mode_np()` can be called before calling `ldap_ssl_environment_init()` or `ldap_ssl_client_init()` to set FIPS processing mode. If FIPS processing mode is supposed to be on, SSL uses the FIPS certified encryption libraries for encryption and sets the processing mode to **on**. FIPS processing mode does not change any existing SSL environments.

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 100.

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.

`ldap_ssl_set_fips_mode_np()` returns `LDAP_SUCCESS` if the client library supports SSL, otherwise it returns `LDAP_SSL_NOT_AVAILABLE`.

See also

`ldap_init`, `ldap_ssl_environment_init`, `ldap_ssl_client_init`

LDAP_START_TLS

`ldap_start_tls_s_np`

Purpose

Start a TLS session.

Synopsis

```
#include <ldap.h>

int ldap_start_tls_s_np (
    LDAP *ld,
    const char *certificateName)
```

Input parameters

ld Specifies the LDAP pointer used in the `ldap_start_tls_s_np()` call.

certificateName

Specifies the name of the certificate to use. It's the same as the parameter used in the `ldap_ssl_environment_init()` API and may be NULL.

Usage

The `ldap_start_tls_s_np()` API is used to secure a previously unsecured connection. It takes a handle from an existing LDAP connection and the name of the certificate to use. If the command is successful, then communication on the connection will be secure until either the connection is closed or an `ldap_stop_tls_s_np()` call is made.

The secure environment must be initialized, either by calling `ldap_ssl_environment_init` or `ldap_ssl_client_init`, before `ldap_start_tls_s_np()` is called.

Errors

`ldap_start_tls_s_np()` returns `LDAP_SUCCESS` if the call was successful, or an LDAP error if the call was unsuccessful.

If the connection is already secure, either by going against the SSL port or by already establishing a TLS session, then `LDAP_OPERATIONS_ERROR` is returned.

If the secure environment has not been initialized through a call to `ldap_ssl_client_init` or `ldap_ssl_environment_init`, then `LDAP_TLS_CLIENT_INIT_NOT_CALLED` is returned.

If the TLS handshake with the server fails, `LDAP_TLS_HANDSHAKE_FAILED` is returned.

If the server is not configured to allow TLS, then `LDAP_PROTOCOL_ERROR` is returned.

If the GSKit environment was not previously initialized, then `LDAP_SSL_CLIENT_INIT_NOT_CALLED` is returned.

If the server does not support TLS, then `LDAP_REFERRAL` is returned. The referred to server might support TLS.

If the server is configured to do TLS, but is currently unable to establish TLS connections, then `LDAP_UNAVAILABLE` is returned.

See also

`ldap_stop_tls_s_np`, `ldap_ssl_environment_init`, `ldap_ssl_client_init`

LDAP_STOP_TLS

ldap_stop_tls_s_np

Purpose

Abandons an open LDAP connection over TLS.

Synopsis

```
#include <ldap.h>

int ldap_stop_tls_s_np(
    LDAP *ld)
```

Input parameters

ld Specifies the LDAP pointer used in the ldap_start_tls_s_np() call.

Usage

The ldap_stop_tls_s_np() API is used to end the TLS session on a connection.

Note that this call closes the connection to the server.

Errors

ldap_stop_tls_s_np() returns LDAP_SUCCESS if the call was successful, an LDAP error if the call was unsuccessful.

See also

ldap_start_tls_s_np, ldap_ssl_environment_init, ldap_ssl_client_init

LDAP_URL

ldap_is_ldap_url
ldap_url_parse
ldap_free_urldesc
ldap_url_search
ldap_url_search_s
ldap_url_search_st

Purpose

LDAP Uniform Resource Locator 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_url_desc(
    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://[hostname]/dn[?attributes[?scope[?filter]]]
```

where:

- *hostname* 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 nonzero 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_search

LDAP_SSL_ENVIRONMENT_INIT

Purpose

ldap_ssl_environment_init() has the same parameters as ldap_ssl_client_init() 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(
    const char *keydatabase,
    const 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 gsk7ikm 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 gsk7ikm 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>

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 78 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 from `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.

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.

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_keylist()` 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_keylist()` using `ldap_free_sort_keylist()`.

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 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 that 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 the server 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 the server 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, These rules 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 does not simply sort each page. See “Simple paged results of search results” on page 119 for more information.

When chasing referrals, the client application must 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.

Errors

The sort routines return an LDAP error code if they encounter an error parsing the result. See “LDAP_ERROR” on page 81 for a list of the LDAP error codes.

Notes

SortString, sortKeyList, controls, serverControls, and attribute must be freed by the caller.

See also

`ldap_search`, `ldap_parse_result`

Chapter 4. Using gsk7IKM

The following key-management program is provided with the Global Security Kit (GSKit):

- gsk7IKM - 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 Tivoli Directory Server. If you use the IBM Tivoli Directory Server version, you also need to set the LIBPATH environment variable as follows:

```
export LIBPATH=<JAVA_home_directory>/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 gsk7IKM 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 177.

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 gsk7IKM.
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 gsk7IKM Java utility by typing:
gsk7IKM
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 gsk7IKM 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 gsk7IKM 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 gsk7IKM 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 `gsk7IKM` 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 `gsk7IKM` 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 `gsk7IKM` 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 171.

The basic steps to creating a self-signed certificate are:

1. Type `gsk7IKM` 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 `gsk7IKM` 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 `gsk7IKM` 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:

- Integriion 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 `gsk7IKM` 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 Freemail 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 `gsk7IKM` 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. Clear the **Set the certificate as a trusted root** check 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 `gsk7IKM` 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 gsk7IKM program can be used to migrate an existing keyring file, as created with mkkf, to the format used by gsk7IKM.

To migrate a keyring file:

1. Type gsk7IKM 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);
}

ldap_memfree(regID);
ldap_unbind(ld);
return 0;
}
```

Chapter 6. LDAP client plug-in programming reference

The following sections provide information about writing client plug-ins.

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 ARGV 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 `ibmldap.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 `ibmldap.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 190 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 `ibmldap.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, `ibmldap.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 78 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 78 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 nonzero 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>
#include <string.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. 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.

Appendix B. LDAP V3 schema

Use the following sections for information about the LDAP V3 schema.

Dynamic schema

The IBM Tivoli Directory Server Version 6.0 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 C. LDAP distinguished names

Distinguished names (DNs) are used to uniquely identify entries in an LDAP or X.500 directory. DN's 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 DN's 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 DN's 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 DN's 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 D. 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:: IGJlZ21ucyB3aXRoIGEgc3BhY2U=
```

Multiple entries within the same LDIF file are separated by a blank line. Multiple blank lines are considered a logical end-of-file.

LDIF examples

LDIF example: Content

An LDIF content file contains entries that can be loaded to the directory. Here is an example of an LDIF content 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/4AAQSkZJRgABAAAAQABAAD/2wBDABALD
A4MChAODQ4SERATGCgaGBYWGDEjJR0o0jM9PDkzODdASFxOQ
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.

LDIF file: Change types

An LDIF file that contains change types allows you to modify and delete existing directory entries. For example, the following LDIF file entry shows the object class insectopia being added to the existing entry dn= cn=foo, ou=bar using the modify change type:

```
dn: cn=foo, ou=bar
changetype: modify
add: objectclass
objectclass: insectopia
```

For a complete list of change types, see RFC 2849.

LDAP controls

Change type files can also contain LDAP controls. LDAP controls can be used to extend certain LDAP Version 3 operations.

A control must contain a unique object identifier (OID) that identifies the control. Make sure your server supports the control you want to use.

The following example shows the LDAP control syntax. Brackets indicate optional data; only the OID is required.

```
control: <OID> [true||false] [string || :: <64string>]
```

Where:

- *OID* is the OID that identifies the control you want to use.
- *string* is a string that does not include Line Feed, Carriage Return, NULL, colon, space or < symbol.
- *64string* is a base-64 encoded string.

The following example uses the Subtree delete control to delete the ou=Product Development, dc=airius, dc=com entry:

```
dn: ou=Product Development, dc=airius, dc=com
control: 1.2.840.113556.1.4.805 true
changetype: delete
```

When controls are included in an LDIF file, implementations might choose to ignore some or all of them. This might be necessary if the changes described in the LDIF file are being sent on an LDAPv2 connection (LDAPv2 does not support controls), or the particular controls are not supported by the remote server. If the criticality of a control is "true", then the implementation must either include the control, or must not send the operation to a remote server.

See "LDAP controls" on page 78 and Appendix F, "Object Identifiers (OIDs) for extended operations and controls," on page 211 for more information.

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 206 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 Tivoli 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:: V2hhdCBhIGNhcmVmdWwgcmlVhZGVyIH1vd
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 Tivoli Directory Server version 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 4. IANA-defined character sets by platform

Character	Conversion Supported				
	Windows	AIX	Solaris	Linux	HP-UX
ISO-8859-1	X	X	X	X	X

Table 4. IANA-defined character sets by platform (continued)

Character Set Name	Conversion Supported				
	Windows	AIX	Solaris	Linux	HP-UX
ISO-8859-2	X	X	X	X	X
ISO-8859-5	X	X	X	X	X
ISO-8859-6	X	X	X	X	X
ISO-8859-7	X	X	X	X	X
ISO-8859-8	X	X	X	X	X
ISO-8859-9	X	X	X	X	X
ISO-8859-15	NA	X	X		X
IBM437	X	NA	NA		NA
IBM850	X	X	NA		NA
IBM852	X	NA	NA		NA
IBM857	X	NA	NA		NA
IBM862	X	NA	NA		NA
IBM864	X	NA	NA		NA
IBM866	X	NA	NA		NA
IBM869	X	X	NA		NA
IBM1250	X	NA	NA		NA
IBM1251	X	NA	NA		NA
IBM1253	X	NA	NA		NA
IBM1254	X	NA	NA		NA
IBM1255	X	NA	NA		NA
IBM1256	X	NA	NA		NA
TIS-620	X	X	NA		NA
EUC-JP	NA	X	X	X	X
EUC-KR	NA	X	X*		NA
EUC-CN	NA	X	X		NA
EUC-TW	NA	X	X		X
Shift-JIS	X	X	X	X	NA
KSC	X	X	NA		NA
GBK	X	X	X*		NA
Big5	X	X	X		X
GB18030	X	X	X	X	NA
HP15CN					X (with non-GB18030)

* Supported on Solaris 7 and higher only.

The new Chinese character set standard (GB18030) is supported with appropriate patches available from www.sun.com and www.microsoft.com:

Note: On Windows 2000, you must set the environment variable zhCNGB18030=TRUE.

Appendix E. 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 152.
- `ldap_open()`—use `ldap_init()`. See “LDAP_INIT” on page 100.
- `ldap_bind()`—use `ldap_simple_bind()`. See “LDAP_BIND / UNBIND” on page 61.
- `ldap_bind_s()`—use `ldap_simple_bind_s()`. See “LDAP_BIND / UNBIND” on page 61.
- `ldap_result2error()`—use `ldap_parse_result()`. See “LDAP_PARSE_RESULT” on page 120.
- `ldap_perror()`—use `ldap_parse_result()`. See “LDAP_PARSE_RESULT” on page 120.
- `ldap_get_entry_controls_np`—use `ldap_get_entry_controls`. See “LDAP_FIRST_ENTRY, LDAP_FIRST_REFERENCE” on page 92.
- `ldap_parse_reference_np`—use `ldap_parse_reference`. See “LDAP_FIRST_ENTRY, LDAP_FIRST_REFERENCE” on page 92.

Appendix F. Object Identifiers (OIDs) for extended operations and controls

The extended operation and control OIDs in this section are in the root DSE of the IBM Tivoli Directory Server 6.0. In this appendix, each OID is defined and its syntax specified in the following formats:

Extended operations:

Description

Gives a brief description of the extended operation.

Request

OID and syntax for the extended operation request. A request generally sets the requestValue field.

Response

OID and syntax for the extended operation response.

Behavior

How the extended operation behaves; who is enabled to send the extended operation; possible return codes.

Scope *The scope of the extended operation.*

Auditing (if applicable)

How this extended operation is audited.

Controls:

Description

Gives a brief description of the control.

OID *OID for the extended operation.*

Syntax

Syntax for the control.

Behavior

How the control behaves; who is enabled to call the control; possible return codes.

Scope *The scope of the control.*

Auditing (if applicable)

How this control is audited.

OIDs for extended operations

The following table shows OIDs for extended operations. Click on a short name or go to the specified page number for more information about an extended operation's syntax and usage.

Table 5. OIDs for extended operations

Short name	Description	OID assigned
“Account status extended operation” on page 213	This extended operation sends the server a DN of an entry which contains a userPassword attribute, and the server sends back the status of the user account being queried: <ul style="list-style-type: none"> • open • locked • expired 	1.3.18.0.2.12.58
“Attribute type extended operations” on page 214	Retrieve attributes by supported capability: operational, language tag, attribute cache, unique or configuration.	1.3.18.0.2.12.46
“Begin transaction extended operation” on page 216	Begin a Transactional context.	1.3.18.0.2.12.5
“Cascading replication operation extended operation” on page 217	This operation performs the requested action on the server it is issued to and cascades the call to all consumers beneath it in the replication topology.	1.3.18.0.2.12.15
“Clear log extended operation” on page 242	Request to Clear log file.	1.3.18.0.2.12.20
“Control replication extended operation” on page 218	This operation is used to force immediate replication, suspend replication, or resume replication by a supplier. This operation is allowed only when the client has update authority to the replication agreement	1.3.18.0.2.12.16
“Control queue extended operation” on page 220	This operation marks items as “already replicated” for a specified agreement. This operation is allowed only when the client has update authority to the replication agreement.	1.3.18.0.2.12.17
“DN normalization extended operation” on page 221	Request to normalize a DN or a sequence of DNs.	1.3.18.0.2.12.30
“Dynamic server trace extended operation” on page 222	Activate or deactivate tracing in the IBM Tivoli Directory Server.	1.3.18.0.2.12.40
“Dynamic update requests extended operation” on page 223	Request to update server configuration for IBM Tivoli Directory Server.	1.3.18.0.2.12.28
“End transaction extended operation” on page 224	End Transactional context (commit/rollback).	1.3.18.0.2.12.6
“Event notification register request extended operation” on page 225	Request registration for events notification.	1.3.18.0.2.12.1
“Event notification unregister request extended operation” on page 226	Unregister for events that were registered for using an Event Registration Request.	1.3.18.0.2.12.3
“Get lines extended operation” on page 243	Request to get lines from a log file.	1.3.18.0.2.12.22
“Get number of lines extended operation” on page 244	Request number of lines in a log file.	1.3.18.0.2.12.24
“Group evaluation extended operation” on page 227	Requests all the groups that a given user belongs to.	1.3.18.0.2.12.50

Table 5. OIDs for extended operations (continued)

Short name	Description	OID assigned
“Kill connection extended operation” on page 228	Request to kill connections on the server. The request can be to kill all connections or kill connections by bound DN, IP, or a bound DN from a particular IP.	1.3.18.0.2.12.35
“LDAP trace facility extended operation” on page 229	Use this extended operation to control LDAP Trace Facility remotely using the Administration Daemon.	1.3.18.0.2.12.41
“Quiesce or unquiesce replication context extended operation” on page 230	This operation puts the subtree into a state where it does not accept client updates (or terminates this state), except for updates from clients authenticated as directory administrators where the Server Administration control is present.	1.3.18.0.2.12.19
“Replication error log extended operation” on page 232	Maintenance of a replication error table.	1.3.18.0.2.12.56
“Replication topology extended operation” on page 233	Trigger a replication of replication topology-related entries under a given replication context.	1.3.18.0.2.12.54
“Start, stop server extended operations” on page 234	Request to start, stop or restart an LDAP server.	1.3.18.0.2.12.26
“Start TLS extended operation” on page 235	Request to start Transport Layer Security.	1.3.6.1.4.1.1466.20037
“Unique attributes extended operation” on page 236	The unique attributes extended operation provides a list of all non-unique (duplicate) values for a particular attribute.	1.3.18.0.2.12.44
“Update configuration extended operation” on page 237	Request to update server configuration for IBM Tivoli Directory Server.	1.3.18.0.2.12.28
“Update event notification extended operation” on page 239	Request that the event notification plug-in get the updated configuration from the server.	1.3.18.0.2.12.31
“Update log access extended operation” on page 239	Request that the log access plug-in get the updated configuration from the server.	1.3.18.0.2.12.32
“User type extended operation” on page 240	Request to get the User Type of the bound user.	1.3.18.0.2.12.37

Account status extended operation

Description

This extended operation sends the server a DN of an entry which contains a userPassword attribute, and the server sends back the status of the user account being queried:

- open
- locked
- expired

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.58

Syntax

```
SEQUENCE {
    dn      LDAPDN
}
```

Response

OID 1.3.18.0.2.12.59

Syntax

```
SEQUENCE {
    status  INTEGER{open(0), locked(1), expired(2)};
}
```

Behavior

This extended operation requests the account status of a user account. The DN is the DN of the user account that is being queried. The server sends back the status of the user account being queried.

The following are enabled to call the extended operation:

- Local Administrators
- Local Administration Group members
- Global Administration Group members

Note: If the extended operation is called by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This extended operation has the following possible return codes:

- LDAP_SUCCESS
- LDAP_NO_MEMORY
- LDAP_OPERATIONS_ERROR
- LDAP_NO_RESULTS_RETURNED
- LDAP_PROTOCOL_ERROR

This extended operation is not supported by the Administration Daemon.

Scope The extended operation only affects the current operation.

Attribute type extended operations

Description

The server needs to provide a way for LDAP clients to determine which attributes in the schema are operational. In addition, there are other attribute characteristics that are being added by new features that cannot be determined by looking at the schema. Currently, the extended operation provides a way for LDAP clients to query about the following attributes:

- Operational - The operational attributes of the server.
- Language Tag - The attributes that can use language tags.
- Attribute Cache - The attributes that can be attribute cached.
- Unique - The attributes that can be marked as unique.
- Configuration - The configuration attributes of the server.

Request

OID 1.3.18.0.2.12.46

Syntax

```
RequestValue ::= SEQUENCE {
    AttributeTypeRequest ENUMERATED {
        OPERATIONAL (0),
        LANGUAGE TAG (1),
        ATTRIBUTE CACHE (2),
        UNIQUE (3),
        CONFIGURATION (4)
    },
    hasCharacteristic BOOLEAN }

```

The extended operation request value takes two parameter on the request. The first parameter is an enumeration telling the server which attribute type (characteristic) is being requested. This release supports queries about the following attributes:

- Operational
- Language Tag
- Attribute Cache
- Unique

The second parameter is a Boolean value that determines whether to return the attributes that have the specified attribute characteristic. A value of FALSE returns a list of attribute names that do not fall into the specified attribute category. A value of TRUE returns a list of attribute names that do fall into the specified attribute category.

Response

OID 1.3.18.0.2.12.47

Syntax

```
ResponseValue ::= SEQUENCE of AttributeNames (or LDAPString);

```

Behavior

This extended operation enables the user to do the following:

- Retrieve a list of all operational attributes
- Retrieve a list of all attributes that can use language tags (not a list of attributes that are using language tags)
- Retrieve a list of all attributes that can be cached (not a list of attributes that are being cached)
- Retrieve a list of all attributes that can made into a unique attribute (not a list of attributes that are currently unique attributes)
- Allows the user to retrieve a list of all attributes that are configuration attributes. These are attributes defined in the configuration schema.

This extended operation also provides an option to return the inverse of any attribute characteristic for which the user queries. For example, the user must be able to ask for all attributes that are not operational attributes.

All user types, including anonymous users, are enabled to call this extended operation. This extended operation has the following possible return codes:

- LDAP_SUCCESS
- LDAP_NO_MEMORY
- LDAP_OTHER
- LDAP_PROTOCOL_ERROR

- LDAP_OPERATIONS_ERROR

This extended operation is not supported by the Administration Daemon.

Scope No effect after the call. It is just a lookup extended operation.

Auditing

AttributeType: *<Type>*

Where *<Type>* is one of the following:

- Operational
- Language Tag
- Attribute Cache
- Unique Attribute
- Configuration

HasCharacteristic: *<Boolean>*

Where *<Boolean>* is one of the following:

- FALSE
- TRUE

Begin transaction extended operation

Description

The Begin transaction extended operation requests that the server start a transactional context on the connection.

Note: This extended operation is enabled by default, but can be disabled by changing the value in the configuration file for the `ibm-slapdTransactionEnable` attribute.

The `ibm-slapdTransactionEnable` attribute is in the configuration file in the `cn=Transaction,cn=configuration` entry. If the value is set to `FALSE`, transactions are not enabled. If set to `TRUE`, transactions are enabled. Transactions can also be enabled or disabled using the Web Administration tool.

Request

OID 1.3.18.0.2.12.5

Syntax

There is no request value.

Response

OID 1.3.18.0.2.12.5

Syntax

```
SEQUENCE {
    transactionID INTEGER
}
```

Behavior

This extended operation puts the connection in the transaction state.

All users can perform this extended operation.

This extended operation has the following possible return codes:

- LDAP_SUCCESS

- LDAP_NO_MEMORY
- LDAP_OPERATIONS_ERROR
- LDAP_UNWILLING_TO_PERFORM

This extended operation is not supported by the Administration Daemon.

Scope This extended operation changes the state of the connection for future operations. This connection remains in the transaction state until a stop transaction extended operation is sent, or an error occurs.

Cascading replication operation extended operation

Description

Perform a replication extended operation on every server in the full replication topology.

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.15

Syntax

```
requestValue ::=SEQUENCE {
  action ActionValue,
  subtreeDN DistinguishedName,
  timeout INTEGER
}
ActionValue ::=INTEGER {
  quiesce (0),
  unquiesce (1),
  replicateNow (2),
  waitForReplication (3)
}
```

Response

OID 1.3.18.0.2.12.15

Syntax

There is no return value.

The following are possible return codes:

- LDAP_SUCCESS
- LDAP_NO_SUCH_OBJECT
- LDAP_UNWILLING_TO_PERFORM
- LDAP_NO_MEMORY
- LDAP_OPERATIONS_ERROR
- LDAP_INSUFFICIENT_ACCESS
- LDAP_PARAM_ERROR
- LDAP_ENCODING_ERROR
- LDAP_LOCAL_ERROR
- LDAP_TIMEOUT

Behavior

The requested operation is performed on the target server and on all replicas of the target server.

The following are enabled to call the extended operation:

- Local Administrators

- Local Administration Group members
- Global Administration Group members
- Master Server DN
- Authenticated Directory User

Note: If the extended operation is called by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This extended operation has the following possible return codes:

- LDAP_SUCCESS
- LDAP_PROTOCOL_ERROR
- LDAP_NO_MEMORY
- LDAP_DECODING_ERROR
- LDAP_UNDEFINED_TYPE
- LDAP_INVALID_DN_SYNTAX

This extended operation is not supported by the Administration Daemon.

Scope This extended operation only affects the current operation.

Auditing

Action: [Quiesce | Unquiesce | ReplNow | Wait | Unknown]

Context DN: <context DN>

Timeout: <timeout>

Control replication extended operation

Description

This extended operation is used to control the following aspects of currently-running replications:

- Suspend replication
- Resume replication
- Cause changes to be replicated immediately

Request

OID 1.3.18.0.2.12.16

Syntax

```
requestValue ::=SEQUENCE {
  action ActionValue,
  scope ScopeValue
  entryDN DistinguishedName
}
ActionValue ::=INTEGER {
  suspend (0),
  resume (1),
  replicateNow (2),
  terminateFullReplication (3)
}
ScopeValue ::=INTEGER {
  singleAgreement (0),
  allAgreements (1)
}
```

Response

OID 1.3.18.0.2.12.16

Syntax

```
Response Value ::=SEQUENCE {  
#fields of interest from LDAPResult:  
resultCode INTEGER (0..MAX),  
errorMessage LDAPString,  
consumer LDAPString  
}
```

The following are possible return codes:

- LDAP_SUCCESS
- LDAP_NO_SUCH_OBJECT
- LDAP_UNWILLING_TO_PERFORM
- LDAP_OPERATIONS_ERROR
- LDAP_INSUFFICIENT_ACCESS
- LDAP_NO_MEMORY

Behavior

This extended operation is used to control the following aspects of currently-running replications:

Suspend replication

Changes are not replicated for the replication agreement or for all replication agreements for the context until the resume replication or replicate immediately operation is used.

Resume replication

If the replication agreement is suspended, then replication resumes.

Cause changes to be replicated immediately

If the replication agreement is suspended or is waiting for scheduled replication to occur, any outstanding changes are replicated.

The following are enabled to call the extended operation:

- Local Administrators
- Local Administration Group members
- Global Administration Group members
- Master Server DN
- Authenticated Directory User

Note: If the extended operation is called by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This extended operation has the following possible return codes:

- LDAP_SUCCESS
- LDAP_PROTOCOL_ERROR
- LDAP_DECODING_ERROR
- LDAP_NO_MEMORY
- LDAP_UNDEFINED_TYPE
- LDAP_INVALID_DN_SYNTAX

This extended operation is not supported by the Administration Daemon.

Scope This extended operation only affects the current operation.

Auditing

Action: [Suspend | Resume | Rep1Now | Unknown]
Scope: [Single | All | Unknown]
DN: <dn>

Control queue extended operation

Description

This extended operation is used to skip changes in the replication queue for an agreement.

Request

OID 1.3.18.0.2.12.17

Syntax

```
requestValue ::=SEQUENCE {  
  action ActionValue,  
  agreementDN DistinguishedName,  
  changeId LDAPString  
}  
ActionValue ::=INTEGER {  
  skipAll (0),  
  skipSingle (1)  
}
```

Response

OID 1.3.18.0.2.12.17

Syntax

```
Response Value ::=SEQUENCE {  
  #fields of interest from LDAPResult:  
  resultCode INTEGER (0..MAX),  
  errorMessage LDAPString,  
  #operation information:  
  changesSkipped INTEGER (0..MAX)  
}
```

The following are possible return codes:

- LDAP_SUCCESS
- LDAP_NO_SUCH_OBJECT
- LDAP_UNWILLING_TO_PERFORM
- LDAP_OPERATIONS_ERROR
- LDAP_INSUFFICIENT_ACCESS
- LDAP_NO_MEMORY

Behavior

This extended operation skips changes in the replication agreements queue. If skipSingle is used, and changeID is the next ID in the replication agreements queue, then changeID is skipped over. If changeID is not at the head of the list of pending changes, the operation fails. If skipAll is used, then all outstanding changes in the replication agreements queue are skipped.

The following are enabled to call the extended operation:

- Local Administrators
- Local Administration Group members
- Global Administration Group members
- Master Server DN

- Authenticated Directory User

Note: If the extended operation is called by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This extended operation has the following possible return codes:

- LDAP_SUCCESS
- LDAP_PROTOCOL_ERROR
- LDAP_DECODING_ERROR
- LDAP_NO_MEMORY
- LDAP_UNDEFINED_TYPE
- LDAP_INVALID_DN_SYNTAX

This extended operation is not supported by the Administration Daemon.

Scope This extended operation only affects the current operation.

Auditing

Skip: [All | <changeId> | Unknown]
 Agreement DN: <agreementDn>

DN normalization extended operation

Description

The DN normalization extended operation normalizes a DN or a list of DNs. The normalization is based on the server’s schema.

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.30

Syntax

```
RequestValue ::= SEQUENCE {
  case INTEGER {preserve(0), normalize (1)};
  SEQUENCE of DistinguishedName;
}
```

Response

OID 1.3.18.0.2.12.30

Syntax

```
ResultValue ::= SEQUENCE {
  SEQUENCE of SEQUENCE {
    Return code INTEGER;
    DN Normalized DistinguishedName;
  }
}
```

Each DN has its own return code. If the return code is not SUCCESS, a DN of zero length is returned for every DN passed in on the original request. The order of DN values in the response matches the order of DN values passed in the request.

LDAP Return Code	Error Condition
Success	The DN was normalized successfully.
UndefinedAttributeType	An attribute in the DN is undefined.
InvalidDNSyntax	The DN syntax is invalid.

Behavior

The extended operation normalizes a DN, or list of DNs. The normalization is based on the schema. See "slapi_dn_normalize_v3" in the *IBM Tivoli Directory Server Plug-ins Reference Version 6.0*.

All users can perform this extended operation.

This extended operation has the following possible return codes:

- LDAP_SUCCESS
- LDAP_PROTOCOL_ERROR
- LDAP_OTHER
- LDAP_OPERATIONS_ERROR

This extended operation is not supported by the Administration Daemon.

Scope The extended operation only affects the current operation.

Dynamic server trace extended operation

Description

Use this extended operation to do the following:

- Start or stop server tracing dynamically
- Set the level of debug data collected
- Name the debug output file

This extended operation depends on the LDAP Trace Facility to be initialized with either the ldtrc command or the successful completion of the LDAP trace facility extended operation request on the IBM Tivoli Directory Server (see "LDAP trace facility extended operation" on page 229).

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.40

Syntax

The value consists of 2 integer values and an optional string. The first integer turns tracing on (1) or off (0). The second integer sets the debug level (0 to 65535) that controls the debug data that is directed to stderr or a file. If the integers are missing, the request fails. If the value is -1, no change is made. The string value provides the file name and is optional. If no name is provided, the name is unchanged. If no name is ever provided, the debug output goes to stderr.

Response

OID 1.3.18.0.2.12.42

Syntax

The response is a string:

```
Trace settings<actual>: enable=%d<%d> trcEvents=%ld<%ld>
  level=0x%x<0x%x> log=[%s]<%s>
```

where values in the brackets show the state after attempting the extended operation. If tracing is on, enable will be 1. The

trcEvents will be 0 if the LDAP Trace Facility is not enabled. Non-zero values indicate that the server was successful in attaching to the LDAP Trace Facility's shared memory buffer. The debug level is shown in hex. The log values is the name of the file used to collect the debug output. It might show stderr if the output is going to the console.

Behavior

This extended operation changes the global variables used to control debugging and tracing in the server. If trace is enabled but the debug level is 0, trace data (function entry and exit points and so forth) is captured in shared memory and nothing is written to the debug file or stderr. If the debug level is between 0 and 65535, different levels of debug data are output to the debug file or stderr. If the LDAP Trace facility is not initialized, no trace output is captured and no debug output is written.

The following are enabled to call the extended operation:

- Local Administrators
- Local Administration Group member

Note: If the extended operation is called by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This extended operation has the following possible return codes:

- LDAP_SUCCESS
- LDAP_INSUFFICIENT_ACCESS
- LDAP_PROTOCOL_ERROR

This extended operation is not supported by the Administration Daemon.

Scope Only the current server session is affected by this operation.

Auditing

The additional information in the audit log is:

Trace=%d [1=on|0=off] debug=0x%x log=[%s]

Dynamic update requests extended operation

Description

The Dynamic update extended operation requests that the server reread its configuration.

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.28

Syntax

```
RequestValue ::= SEQUENCE {
    action INTEGER {rereadFile(0),
                  rereadAttribute(1),
                  rereadEntry(2),
                  rereadSubtree(3)};
    entry [0] DistinguishedName OPTIONAL;
    attribute [1] DirectoryString OPTIONAL;
}
```

Response

OID 1.3.18.0.2.12.29

Syntax

There is no response value.

Behavior

This extended operation forces the server to reread the configuration file. The request can be to reread the entire file, a subtree, an entry or a specific attribute. When the server receives the request, the server rereads the configuration file and updates all the internal server settings to use the new settings from the configuration file. Only the dynamic attributes are reread.

Only the local Administrator or Local Administration Group members are enabled to call this extended operation. Local Administration Group members cannot update attributes of other Local Administration Group members.

Note: If the control is sent by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This control has the following possible return codes:

- LDAP_SUCCESS
- LDAP_UNDEFINED_TYPE
- LDAP_INSUFFICIENT_ACCESS
- LDAP_INVALID_SYNTAX
- LDAP_INVALID_DN_SYNTAX
- LDAP_UNWILLING_TO_PERFORM
- LDAP_OBJECT_CLASS_VIOLATION
- LDAP_OTHER
- LDAP_PROTOCOL_ERROR
- LDAP_NO_SUCH_ATTRIBUTE
- LDAP_NO_SUCH_OBJECT
- LDAP_NO_MEMORY

This extended operation is not supported by the Administration Daemon.

Scope This extended operation causes the server to reread its configuration, which can affect subsequent operations.

Auditing

Scope: *<Scope Value>*

where *<Scope Value>* can be one of the following:

- Entire - entire configuration file
- Single - for a single attribute
- Entry - for an entry
- Subtree - for a subtree

DN: *<Entry DN>* – This is required for Single, Entry and Subtree.

Attribute: *<Attribute>* – This is required for Single only.

End transaction extended operation

Description

The End transaction extended operation requests that the server commit all

the operations performed inside the transaction and change the state of the connection so it is no longer in the transactional state.

Note: This extended operation is enabled by default, but can be disabled by changing the value in the configuration file for the `ibm-slapdTransactionEnable` attribute.

The `ibm-slapdTransactoinEnabled` attribute is in the configuration file in the `cn=Transaction,cn=configuration` entry. If the value is set to `FALSE`, transactions are not enabled. If set to `TRUE`, transactions are enabled. Transactions can also be enabled or disabled using the Web Administration tool.

Request

OID 1.3.18.0.2.12.6

Syntax

```
SEQUENCE {
    transactionVote ENUMERATED {
        commit (0),
        rollback(1)
    },
    transactionID INTEGER
}
```

Response

OID 1.3.18.0.2.12.6

Syntax

```
SEQUENCE {
    transactionID INTEGER
}
```

Behavior

The extended operation commits the transaction and removes the connection from the transaction state.

All users can perform this extended operation.

This extended operation has the following possible return codes:

- LDAP_SUCCESS
- LDAP_NO_MEMORY
- LDAP_OPERATIONS_ERROR
- LDAP_UNWILLING_TO_PERFORM
- LDAP_TIMELIMIT_EXCEEDED
- LDAP_SIZELIMIT_EXCEEDED

This extended operation is not supported by the Administration Daemon.

Scope This extended operation changes the state of the connection for future operations. The extended operation takes the connection out of the transactional state.

Event notification register request extended operation

Description

The operation allows a client to request that the server notify the client when a portion of the tree has changed.

Note: Event notification can be turned off by setting the attribute `ibm-slapdEnableEventNotification` in the entry `cn=Event Notification, cn=Configuration` to `FALSE`.

Request

OID 1.3.18.0.2.12.1

Syntax

```
changeType ::= ENUMERATED {
    changeAdd (1),
    changeDelete (2),
    changeModify (4),
    changeModDN (8) }
requestValue = SEQUENCE {
    eventID ENUMERATED {
        LDAP_CHANGE (0)},
    baseObject LDAPDN,
    scope ENUMERATED {
        baseObject (0),
        singleLevel (1),
        wholeSubtree (2) },
    type INTEGER OPTIONAL }
```

Response

OID 1.3.18.0.2.12.1

Syntax

```
response ::= OCTET STRING
```

Behavior

If successful, the server sends an unsolicited notification to the client when a modification happens that the client is interested in.

All users other than anonymous can perform this extended operation.

This extended operation has the following possible return codes:

- `LDAP_UNWILLING_TO_PERFORM`
- `LDAP_NO_SUCH_OBJECT`
- `LDAP_UNDEFINED_TYPE`

This extended operation is not supported by the Administration Daemon.

Scope If successful, the client may receive unsolicited notifications from the server.

Auditing

```
eventID: LDAP_change
base: baseDn
scope: baseObject, singleLevel, or wholeSubtree
```

Event notification unregister request extended operation

Description

The operation allows a client to request that the server stop notifying the client when a portion of the tree has changed.

Note: Event notification can be turned off by setting the attribute `ibm-slapdEnableEventNotification` in the entry `cn=Event Notification, cn=Configuration` to `FALSE`.

Request

OID 1.3.18.0.2.12.3

Syntax

```
requestValue ::= OCTET STRING
```

Response

OID 1.3.18.0.2.12.4

Syntax

If the registration is successfully removed, the LDAPResult field contains LDAP_SUCCESS and the response field contains the registration ID that was removed.

Behavior

If successful, the server will stop sending unsolicited notifications to the client when a modification happens that the client was interested in.

All users other than anonymous can perform this extended operation.

This extended operation has the following possible return codes:

- LDAP_UNWILLING_TO_PERFORM
- LDAP_NO_SUCH_OBJECT
- LDAP_UNDEFINED_TYPE

This extended operation is not supported by the Administration Daemon.

Scope If successful, the client will stop receiving unsolicited notifications from the server.

Auditing

ID: hostname.uuid

Group evaluation extended operation

Description

The Group evaluation extended operation requests that the server return the set of groups to which the requested user belongs.

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.50

Syntax

```
GroupEvaluationRequestValue:: = SEQUENCE {
    dn LDAPDN,
    attributes AttributeList OPTIONAL
}
```

Response

OID 1.3.18.0.2.12.52

Syntax

```
Group ::= SEQUENCE { groupName LDAPString }
GroupEvaluationResponseValue:: = SEQUENCE {
    normalized INTEGER{unnormalized(0), normalized(1)};
    Sequence of Group }
```

Behavior

This extended operation determines to which groups the requested user belongs.

The following are enabled to call the extended operation:

- Local Administrators

- Local Administration Group members
- Global Administrators

Note: If the extended operation is called by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This control has the following possible return codes:

- LDAP_SUCCESS
- LDAP_NO_MEMORY
- LDAP_OPERATIONS_ERROR
- LDAP_INVALID_DN_SYNTAX
- LDAP_NO_RESULTS_RETURNED
- LDAP_PROTOCOL_ERROR
- LDAP_NO_SUCH_ATTRIBUTE

This extended operation is not supported by the Administration Daemon.

Scope The extended operation only affects the current operation.

Auditing

The group evaluation extended operation sets the audit string to

DN: <the DN sent in the group evaluation extended operation> \n

If `ibm-auditAttributesOnGroupEvalOp` is TRUE, the audit string will contain a list of attribute value pairs separated by a new line. If the `ibm-auditAttributesOnGroupEvalOp` is FALSE, the string will contain:

sentAttrs: <true|false>

The value will be FALSE if no attributes were sent on the request

Kill connection extended operation

Description

The Kill connection extended operation requests that the server stop the specified connections. Connections can be stopped based on the following:

- Connection IP
- Connection DN
- Combination of IP and DN
- All connections

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.35

Syntax

```
ReqType ::= ENUMERATED {
  DN (1),
  IP (2)
}
RequestValue ::= SEQUENCE {
  SET {type ReqType
        value Directory String} OPTIONAL
  SET {type ReqType
        value Directory String} OPTIONAL
}
```

For a DN-specific or IP-specific request, only one set of type and value is needed. For a combination DN/IP request, both sets of type and value are needed. If there is no value specified, all connections are stopped.

Response

OID 1.3.18.0.2.12.36

Syntax

```
ResponseValue ::= { int numberKilled  
                    int numberPending }
```

Each DN has its own return code. If the return code is not SUCCESS, a DN of zero length is returned for every DN passed in on the original request. The order of DN values in the response matches the order of DN values passed in the request.

Behavior

This extended operation stops the requested connections.

The following are enabled to call the extended operation:

- Local Administrators
- Local Administration Group members
- Global Administrators

Note: If the extended operation is called by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This extended operation has the following possible return codes:

- LDAP_SUCCESS
- LDAP_NO_SUCH_OBJECT
- LDAP_INSUFFICIENT_ACCESS
- LDAP_INVALID_DN_SYNTAX
- LDAP_OTHER
- LDAP_PROTOCOL_ERROR

This extended operation is not supported by the Administration Daemon.

Scope The extended operation only affects the current operation.

Auditing

The DN or IP or both will be provided:

DN: <DN>

IP: <IP>

If neither the DN nor the IP is present, then the request was to stop all connections.

LDAP trace facility extended operation

Description

Use this extended operation to control LDAP Trace Facility remotely using the Administration Daemon.

Note: This extended operation is always enabled on the Administration Daemon. It is not supported on the Directory Server.

Request

OID 1.3.18.0.2.12.41

Syntax

The value consists of 1 integer value and a string. The first integer has the following values:

- 1– Enables the LDAP Trace Facility
- 2– Disables the LDAP Trace Facility
- 3– Enables changing masks or other parameters
- 4– Clears data already collected in the shared memory buffer
- 5– Shows information about the current state
- 6– Creates a file from the data already captured in shared memory

The optional string contains additional parameters understood by the ldtrc command, such as the size of the buffer (1) or the name of the output file for dump (6).

Response

OID 1.3.18.0.2.12.43

Syntax

The response is a string containing the output from the ldtrc command submitted remotely.

Behavior

The extended operation submits an ldtrc command on the host machine and captures its output to return to the client.

The following are enabled to call the extended operation:

- Local Administrators
- Local Administration Group member

Note: If the extended operation is called by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This extended operation has the following possible return codes:

- LDAP_SUCCESS
- LDAP_INSUFFICIENT_ACCESS
- LDAP_PROTOCOL_ERROR

This extended operation is supported by the Administration Daemon only.

Scope The extended operation runs until the machine is rebooted, the root manually issues IPC commands, the ldtrc command is issued on the machine or another request is made.

Auditing

The additional information in the audit log is:

OPTIONS: *<request value><optional string>*

where *<request value>* is the request value (1-6) and *<optional string>* is any additional parameters for ldtrc.

Quiesce or unquiesce replication context extended operation

Description

This extended operation is used for the following:

- Disable non-replication topology-related changes in the replication context.
- Enable non-replication topology-related changes.

Request

OID 1.3.18.0.2.12.19

Syntax

```
requestValue ::=SEQUENCE {
  quiesce BOOLEAN,
  subtreeDn DistinguishedName
}
```

Response

OID 1.3.18.0.2.12.19

Syntax

```
Response Value ::=SEQUENCE {
  #fields of interest from LDAPResult:
  resultCode INTEGER (0..MAX),
  errorMessage LDAPString,
}
```

The following are possible return codes:

- LDAP_SUCCESS
- LDAP_NO_SUCH_OBJECT
- LDAP_UNWILLING_TO_PERFORM
- LDAP_OPERATIONS_ERROR
- LDAP_INSUFFICIENT_ACCESS
- LDAP_NO_MEMORY
- LDAP_REPL QUIESCE_BAD_STATE

Behavior

This extended operation is used for the following:

- Disable non-replication topology-related changes in the replication context.
- Enable non-replication topology-related changes.

If the quiesce Boolean is TRUE, then only replication topology-related changes are enabled.

The following are enabled to call the extended operation:

- Local Administrators
- Local Administration Group members
- Global Administration Group members
- Master Server DN
- Authenticated Directory User

Note: If the extended operation is called by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This extended operation has the following possible return codes:

- LDAP_SUCCESS
- LDAP_PROTOCOL_ERROR
- LDAP_DECODING_ERROR

- LDAP_NO_MEMORY
- LDAP_UNDEFINED_TYPE
- LDAP_INVALID_DN_SYNTAX

This extended operation is not supported by the Administration Daemon.

Scope This extended operation only affects the current operation.

Auditing

Action: [Quiesce | Unquiesce]
Context DN: <dn>

Replication error log extended operation

Description

Use this extended operation to monitor replication errors and correct any problems that occur as data fails to be replicated.

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.56

Syntax

The value consists of an integer which indicates the type of request and two strings in BER format. The first string identifies which failure or failures are to be deleted, attempted again or displayed. The value will either be 0 for all, or the ID of the failed change. The second string provides the DN for the replication agreement.

Response

OID 1.3.18.0.2.12.57

Syntax

The response is a string indicating any problem that occurred, or if successful, how many failed changes were deleted or present to the consumer.

Behavior

The extended operation acts on the table that maintains the updates that failed on any of the current server's consumer servers. The data for any single failure can be displayed. Any or all failed changes can be deleted or attempted again. Deleted changes are removed from the table. Changes attempted again are sent individually to the consumer. If the update succeeds, the failure is removed from the table. If the update fails again, it is added back as a new failure with the number of attempts, last time attempted and result code updated to reflect this. The original failure is removed. The worker thread handling the extended operation connects to the consumer and sends these changes. Replica threads can send updates to the consumer at the same time.

The following are enabled to call the extended operation:

- Local Administrators
- Local Administration Group member
- Users with write access to the replica group

Note: If the extended operation is called by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This extended operation has the following possible return codes:

- LDAP_SUCCESS
- LDAP_INSUFFICIENT_ACCESS
- LDAP_DECODING_ERROR
- LDAP_PROTOCOL_ERROR
- LDAP_UNWILLING_TO_PERFORM
- LDAP_NO_SUCH_OBJECT

This extended operation is not supported by the Administration Daemon.

Scope If the errors are deleted or successfully attempted again, they are removed from the table permanently.

Auditing

The additional information in the audit log is consists of three lines:

Replication Error Log Management Option: [SHOW | RETRY | DELETE | UNKNOWN]

Replication Error ID: <numeric value>

Replication Agreement DN: DN or empty string.

Replication topology extended operation

Description

This extended operation propagates replication topology-related entries from a supplier to the consumers in the network. This extended operation is useful to synchronize replication topology data for every server in the network before replication of directory entries can begin.

Request

OID 1.3.18.2.12.54

Syntax

```
RequestValue ::= SEQUENCE {
    replicationContextDn DistinguishedName,
    timeout INTERGER,
    replicationAgreementDn DistinguishedName OPTIONAL
}
```

Response

OID 1.3.18.0.2.12.55

Syntax

```
ResponseValue ::= SEQUENCE {
    resultCode INTEGER(0..MAX),
    errorMessage LDAPString,
    #operation specific failure information:
    supplier LDAPString,
    consumer LDAPString,
}
```

Behavior

A supplier gathers its replication topology-related entries under a replication context and propagates them to the consumer servers. The supplier can add the entries to the consumer or modify the existing entries on the consumer or delete the extra entries from the consumer. As a result of the extended operation, the replication topology related entries under the specified context on both the supplier and the consumers are in sync.

The operation is enabled when the client is authenticated with update authority to all agreements in the specified subtree, or is authenticated as a master server for the specified subtree.

Note: If the extended operation is called by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This extended operation has the following possible return codes:

- LDAP_SUCCESS
- LDAP_NO_MEMORY
- LDAP_OPERATIONS_ERROR
- LDAP_UNWILLING_TO_PERFORM
- LDAP_PROTOCOL_ERROR

This extended operation is not supported by the Administration Daemon.

Scope The extended operation will not affect subsequent operation on the connection.

Auditing

Context DN, Replication Agreement DN and Timeout are audited.

Start, stop server extended operations

Description

The Start, stop server extended operation, when sent to the Administration Daemon, requests that the Administration Daemon start, stop, restart, give the status of the LDAP server, or stop the Administration Daemon. The Start Stop Server Extended Operation, when sent to the LDAP Server, requests that the LDAP Server stop.

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.26

Syntax

```
actionType ::= ENUMERATED {
    startServer (0),
    stopServer (1),
    restartServer (2),
    serverStatus (3),
    admStop (4)}

requestValue ::= SEQUENCE {
    action actionType
    command options string OPTIONAL
}
```

Response

OID 1.3.18.0.2.12.27

Syntax

```
ResultValue ::= SEQUENCE {
    Status Integer
    ErrorString String
}
```

Behavior

When sent to the Administration Daemon, the request does one of the following:

- Start
- Restart
- Stop

- Request the server's status
- Stop the Administration Daemon

When sent to the LDAP Server, the server only honors the request to stop the server. Any other request sent to the LDAP Server results in a return code of LDAP_UNWILLING_TO_PERFORM.

When the request is sent to the Administration Daemon, only a local admin or local admin group member has the authority to make the request.

When the request is sent to the LDAP server, a local admin, local admin group member or a global admin group member has the authority to make the request.

This control has the following possible return codes:

- LDAP_SUCCESS
- LDAP_OTHER
- LDAP_UNWILLING_TO_PERFORM
- LDAP_INSUFFICIENT_ACCESS
- LDAP_PROTOCOL_ERROR

This extended operation is supported by the Administration Daemon. This extended operation with the stop request is supported in the LDAP Server.

Scope The extended operation only affects the current operation, unless the request is to stop the admin daemon.

Auditing

In the LDAP server, the additional information contains:

Operation: <Start | Stop | Restart | Admin Stop | Status>

In the Administration Daemon, the additional information contains:

Operation: <Start | Stop | Restart | Admin Stop | Status>

On a start or restart operation the following line is audited:

Options: <Additional Value>

For example, a request to start the server with the **-a** option audits the following:

Operation: Start

Options: ---a

Start TLS extended operation

Description

This extended operation requests that the server start using encrypted communications over the connection.

Note: This extended operation is always enabled.

Request

OID 1.3.6.1.4.1.1466.20037

Syntax

There is no request value for the extended operation.

Response

OID 1.3.6.1.4.1.1466.20037

Syntax

- LDAP_SUCCESS
- LDAP_OPERATIONS_ERROR
- LDAP_PROTOCOL_ERROR

Behavior

The extended operation is used to request that communication on the connection be encrypted. The server will expect a TLS handshake on the connection.

All users can perform this extended operation.

Note: If the extended operation is called by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This extended operation has the following possible return codes:

- LDAP_SUCCESS
- LDAP_OPERATIONS_ERROR
- LDAP_PROTOCOL_ERROR

This extended operation is supported by the Administration Daemon.

Scope Once a TLS handshake is performed, all communication on the connection is encrypted until a TLS closure alert is sent or the connection is closed.

Unique attributes extended operation

Description

The unique attributes extended operation provides a list of all non-unique (duplicate) values for a particular attribute.

Note: This extended operation can be disabled. Commenting out or removing the statement in the configuration file for the unique attribute extended operation plug-in will disable this extended operation. For example, commenting out the statement:

```
ibm-slapdPlugin: extendedop /bin/libback-rdbm.dll initUniqueAttr
```

from the configuration file will disable this extended operation on Windows systems.

Request

OID 1.3.18.0.2.12.44

Syntax

```
ExtendedRequest ::= SEQUENCE {  
    requestName LDAPOID // OID for the IBM Unique Attributes  
    requestValue LDAPOID // OID for an attribute requiring uniqueness  
}
```

where *LDAPOID* is an OCTET STRING.

Response

OID 1.3.18.0.2.12.45

Syntax

```
ExtendedResponse ::= SEQUENCE {  
    COMPONENTS OF LDAPResult,  
    responseName  LDAPOID // OID for the IBM Unique Attributes  
    Response      AttributeValueList // list of all  
                  conflicting attribute values  
}
```

where *AttributeValueList* is a SEQUENCE OF AttributeValue and *LDAPOID* is an OCTET STRING

Behavior

The extended operation lists all non-unique values for a particular attribute.

The following are enabled to call the extended operation:

- Local Administrators
- Local Administration Group members
- Global Administration Group members
- Master Server DN

Note: If the extended operation is called by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This extended operation has the following possible return codes:

- LDAP_SUCCESS
- LDAP_INSUFFICIENT_ACCESS
- LDAP_NO_MEMORY
- LDAP_PARAM_ERROR
- LDAP_OPERATIONS_ERROR
- LDAP_OTHER

This extended operation is not supported by the Administration Daemon.

Scope This extended operation only affects the current operation.

Update configuration extended operation

Description

Request to update server configuration for IBM Tivoli Directory Server.

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.28

Syntax

```
RequestValue ::= SEQUENCE {  
    action  INTEGER {rereadFile(0),  
                   rereadAttribute(1),  
                   rereadEntry(2),  
                   rereadSubtree(3)};  
    entry   [0] DistinguishedName OPTIONAL;  
    attribute [1] DirectoryString OPTIONAL;  
}
```

Response

OID 1.3.18.0.2.12.29

Syntax

This response has no value.

Behavior

This extended operation forces the server to read the configuration file. The request can be to read the entire file, a sub-tree, an entry or a specific attribute. When the server receives the request it reads the configuration file and updates all the internal server settings to use the new settings from the configuration file. Only those attributes which are dynamic are read.

The following are enabled to call the extended operation:

- Local Administrators
- Local Administration Group member

Notes:

1. Local Administration Group members cannot update other Local Administration Group member's attributes.
2. If the extended operation is called by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This extended operation has the following possible return codes:

- LDAP_SUCCESS
- LDAP_INSUFFICIENT_ACCESS
- LDAP_DECODING_ERROR
- LDAP_PROTOCOL_ERROR
- LDAP_UNWILLING_TO_PERFORM
- LDAP_NO_SUCH_OBJECT
- LDAP_UNDEFINED_TYPE
- LDAP_INVALID_SYNTAX
- LDAP_INVALID_DN_SYNTAX
- LDAP_OBJECT_CLASS_VIOLATION
- LDAP_NO_SUCH_ATTRIBUTE
- LDAP_NO_MEMORY

This extended operation is not supported by the Administration Daemon.

Scope This extended operation causes the server to read its configuration, which might affect subsequent operations.

Auditing

The Scope will be provided along with the entry dn, and/or attribute when necessary:

Scope: <Scope Value>

Where *Scope Value* is one of the following:

- Entire - entire configuration file
- Single - for a single attribute
- Entry - for an entry
- Subtree - for a subtree

DN: <Entry DN> * this is required for Single, Entry and subtree

Attribute: <Attribute> *this is required for single only.

Update event notification extended operation

Description

The Update event notification extended operation requests that the event notification plug-in get the updated configuration from the server. This operation can only be requested by the server. A Client application cannot request this operation. The operation is initiated by the server after receiving a dynamic update request that affects the event notification plug-in.

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.31

Syntax

There is no request value for the extended operation.

Response

OID 1.3.18.0.2.12.31

Syntax

There is no response value for the extended operation.

Behavior

The extended operation forces the event notification plug-in to get the maximum events and maximum events per connection settings from the server using the global pblock.

Only the server or an internal server plug-in are enabled to call this extended operation.

Note: If the control is sent by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This control has the following possible return codes:

- LDAP_SUCCESS
- LDAP_INSUFFICIENT_ACCESS
- LDAP_NO_MEMORY

This extended operation is not supported by the Administration Daemon.

Scope This extended operation changes the event settings which can affect all subsequent operations.

Update log access extended operation

Description

The Update log access extended operation requests that the log access plug-in get the updated configuration from the server. This operation can only be requested by the server. A Client application cannot request this operation. The operation is initiated by the server after receiving a dynamic update request that affects the log Access plug-in.

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.32

Syntax

There is no request value for the extended operation.

Response

OID 1.3.18.0.2.12.32

Syntax

There is no response value for the extended operation.

Behavior

The extended operation forces the log access plug-in to get the latest log file locations from the server.

Only the server or an internal server plug-in are enabled to call this extended operation.

Note: If the control is sent by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This control has the following possible return codes:

- LDAP_SUCCESS
- LDAP_INSUFFICIENT_ACCESS
- LDAP_NO_MEMORY

This extended operation is not supported by the Administration Daemon.

Scope This extended operation changes the log access settings which can affect all subsequent log access operations.

User type extended operation

Description

This extended operation can be used by a bound user to determine the user type and roles the user has on the IBM Tivoli Directory Server. Without the extended operation, there is no programmatic way to determine what general capabilities a user has and where the DN and password for the user are stored.

It is possible for a user to belong to a user type and have different capabilities and store passwords under different types of entries or attributes.

Additionally, the extended operation provides a way to distinguish the root administrator from an administrative group member when a client must use the Administration Daemon to authenticate a user.

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.37

Syntax

There is no request value for the extended operation.

Response

OID 1.3.18.0.2.12.38

Syntax

```
ResponseValue ::= SEQUENCE
{
    STRING (UserType)
    INTEGER (Number of UserRoles)
    SEQUENCE OPTIONAL
    {
        STRING (UserRole)
    }
}
```

Behavior

This extended operation can be used by a bound user to determine the user type and roles the user has on the IBM Tivoli Directory Server.

All users, including anonymous, are enabled to send the control.

This control has the following possible return codes:

- LDAP_SUCCESS
- LDAP_NO_MEMORY
- LDAP_OPERATIONS_ERROR
- LDAP_NO_RESULTS_RETURNED
- LDAP_PROTOCOL_ERROR

This extended operation is not supported by the Administration Daemon.

Scope The extended operation only affects the current operation.

Log access extended operations

Three types of extended operation requests support access to the log files. The IBM Tivoli Directory Server administrator supports the following log access extended operations:

- “Clear log extended operation” on page 242
- “Get lines extended operation” on page 243
- “Get number of lines extended operation” on page 244

The server provides access to the following log files:

- ibmslapd.log
- db2cli.log
- db2clicmds.log
- audit.log
- bulkload.log
- ibmdiradm.log
- lostandfound.log
- idstools.log
- db2load.log
- tracemsg.log
- adminAudit.log (this file is available only if the Administration Daemon audit log OID (1.3.18.0.2.32.11) is in the list of supported capabilities in the root DSE)
- ibmslapd.trace.log (this file is available only if the trace log OID (1.3.18.0.2.32.14) is in the list of supported capabilities in the root DSE)

Lines are numbered starting with line 0. A line is considered all characters up to and including a new line or 400 characters, whichever comes first.

To make the log access request, a client application can use the 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 }
```

All the extended requests use a LogType. LogType is defined as:

```
LogType ::= ENUMERATED {
    SlapdErrors (1),
    CLIErrors (2),
    AuditLog (4),
    BulkloadLog (8),
    AdminErrors (16),
    AdminAudit (32),
    DebugOutputFile (64),
    LostAndFound (128),
    ConfigToolsLog (256)}
RequestValue ::= { log LogType; }
```

Clear log extended operation

Description

The Clear log extended operation requests that the server clear the requested log. Once the log is cleared a line is written to the log file with the date and time stating that the log file was cleared.

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.20

Syntax

```
RequestValue ::= { log LogType; }
```

Response

OID 1.3.18.0.2.12.21

Syntax

There is no response value.

Behavior

The extended operation clears the requested log file and writes a message in the log with the date and time stating that the log was cleared.

Only the local Administrator or Local Administration Group members are enabled to call this extended operation. Only the local Administrator can clear the audit log. A Local Administration Group member does not have access to clear the audit log.

Note: If the control is sent by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This control has the following possible return codes:

- LDAP_SUCCESS
- LDAP_INSUFFICIENT_ACCESS
- LDAP_UNWILLING_TO_PERFORM
- LDAP_PROTOCOL_ERROR
- LDAP_NO_MEMORY

This extended operation is not supported by the Administration Daemon.

Scope This extended operation only affects the current operation.

Auditing

Log: <Log name>

Get lines extended operation

Description

The Get lines extended operation requests that the server read the specified lines from the requested log and return them to the client.

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.22

Syntax

```
RequestValue ::= SEQUENCE
{
    Log      LogType;
    firstLine INTEGER;
    lastLine INTEGER;
}
```

Response

OID 1.3.18.0.2.12.23

Syntax

There is a response value only if the return code is LDAP_SUCCESS.

Behavior

This extended operation reads the requested set of lines from the requested file and returns the lines to the user.

Only the local Administrator or Local Administration Group members are enabled to call this extended operation.

Note: If the control is sent by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This control has the following possible return codes:

- LDAP_SUCCESS
- LDAP_INSUFFICIENT_ACCESS
- LDAP_UNWILLING_TO_PERFORM
- LDAP_PROTOCOL_ERROR
- LDAP_NO_MEMORY

This extended operation is not supported by the Administration Daemon.

Scope This extended operation only affects the current operation.

Auditing

Log: <Log name>

Get number of lines extended operation

Description

The Get number of lines extended operation requests that the server determine the number of lines in the requested log file.

Note: This extended operation is always enabled.

Request

OID 1.3.18.0.2.12.24

Syntax

```
LogType ::= ENUMERATED {SlapdErrors (1),
    CLIErrors (2),
    AuditLog (4),
    BulkloadLog (8),
    AdminErrors (16),
    AdminAudit (32),
    DebugOutputFile (64),
    LostAndFound (128),
    ConfigToolsLog (256)}
RequestValue ::= { log LogType; }
```

Response

OID 1.3.18.0.2.12.25

Syntax

```
ResponseValue ::= <number of lines>
```

Behavior

The extended requests that the server read the log file and determine the number of lines in the requested log file.

Only the local Administrator or Local Administration Group members are enabled to call this extended operation.

Note: If the control is sent by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This control has the following possible return codes:

- LDAP_SUCCESS
- LDAP_INSUFFICIENT_ACCESS
- LDAP_UNWILLING_TO_PERFORM
- LDAP_PROTOCOL_ERROR
- LDAP_NO_MEMORY

This extended operation is not supported by the Administration Daemon.

Scope This extended operation only affects the current operation.

Auditing

Log: <Log name>

OIDs for controls

The following table shows OIDs for controls. Click on the short name or go the specified page number for more information about a control's syntax and usage.

Table 6. OIDs for controls

Short name	Description	OID assigned
"AES bind control" on page 246	This control enables the IBM Tivoli Directory Server to send updates to the consumer server with passwords already encrypted using AES.	1.3.18.0.2.10.28
"Audit control" on page 247	The control sends a sequence of uniqueid strings and a source ip string to the server. When the server receives the control, it audits the list of uniqueids and sourceip in the audit record of the operation.	1.3.18.0.2.10.22
"Do not replicate control" on page 248	This control can be specified on an update operation (add, delete, modify, modDn, modRdn).	1.3.18.0.2.10.23
"Group authorization control" on page 248	The control sends a list of groups that a user belongs to.	1.3.18.0.2.10.21
"Manage DSAIT control" on page 250	Causes entries with the "ref" attribute to be treated as normal entries, allowing clients to read and modify these entries.	2.16.840.1.113730.3.4.2
	Attached to a delete or modify DN request to cause the server to do only the group referential integrity processing for the delete or rename request without doing the actual delete or rename of the entry itself. The entry named in the delete or modify DN request does not need to exist on the server.	1.3.18.0.2.10.25
"No replication conflict resolution control" on page 251	When present, a replica server accepts a replicated entry without trying to resolve any replication conflict for this entry.	1.3.18.0.2.10.27
"Omit group referential integrity control" on page 252	Omits the group referential integrity processing on a delete or modrdn request. When present on a delete or rename operation, the entry is deleted from or renamed in the directory, but the entry's membership is not removed or renamed in the groups in which the entry is a member.	1.3.18.0.2.10.26
"Paged search results control" on page 252	Allows management of the amount of data returned from a search request.	1.2.840.113556.1.4.319
"Password policy request control" on page 254	Password policy request or response	1.3.6.1.4.1.42.2.27.8.5.1
"Proxy authorization control" on page 255	The Proxy Authorization Control enables a bound user to assert another user's identity. The server uses this asserted identity in the evaluation of ACLs for the operation.	2.16.840.1.113730.3.4.18
"Refresh entry control" on page 256	This control is returned when a target server detects a conflict during a replicated modify operation.	1.3.18.0.2.10.24
"Replication supplier bind control" on page 257	This control is added by the supplier, if the supplier is a gateway server.	1.3.18.0.2.10.18
"Replication update ID control" on page 258	This control was created for serviceability. If the supplier server is set to issue the control, each replicated update is accompanied by this control.	1.3.18.0.2.10.29

Table 6. OIDs for controls (continued)

Short name	Description	OID assigned
“Server administration control” on page 258	Allows an update operation by the administrator under conditions when the operation would normally be refused (server is quiesced, a read-only replica, etc.)	1.3.18.0.2.10.15
“Sorted search results control” on page 259	Allows a client to receive search results sorted by a list of criteria, where each criterion represents a sort key.	1.2.840.113556.1.4.473
“Subtree delete control” on page 260	This control is attached to a Delete request to indicate that the specified entry and all descendent entries are to be deleted.	1.2.840.113556.1.4.805
“Transaction control” on page 261	Marks the operation as part of a transactional context.	1.3.18.0.2.10.5

AES bind control

Description

This control enables the IBM Tivoli Directory Server to send updates to the consumer server with passwords already encrypted using AES. If the consumer server does not support AES encryption of passwords, or the seed or salt values do not match, the IBM Tivoli Directory Server decrypts the userpassword and secretkey values in updates to be replicated.

Note: This control is always enabled.

OID 1.3.18.0.2.10.28

Syntax

This control has no value.

Behavior

The criticality must be set to TRUE in order to protect clients from submitting a request with an unauthorized identity. This control can operate independent of other controls. However, it is often sent with the Proxy Authorization Control. This control is registered for the following operations:

- Bind

The following are enabled to send the control:

- Local Administrators
- Master Server DN
- Local Administration Group members
- Global Administration Group members

Note: If the control is sent by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This control has the following possible return codes:

- LDAP_INSUFFICIENT_ACCESS

This control is not supported by the Administration Daemon.

Scope The control lasts for the life of the bind session, to allow for multiple write operations.

The use of the control implies that cryptographic consistency has been verified by the caller. At bind time the presence of this control, along with the proper authorization, causes the `c_isConsistent` flag in the connection structure to be set to `TRUE`. This causes any write operations containing pre-encrypted AES data to be accepted by the server. Without the presence of the control, the connection flag is set to `FALSE`, and a write operation of this type is rejected by the server. The RDBM backend is the only backend that sets, and evaluates, the `c_isConsistent` flag.

Audit control

Description

The Audit Control enables a client to send additional information on an operation. This additional information is a unique ID and an IP address. The additional information is audited in the audit log.

Note: This control is always enabled.

OID 1.3.18.0.2.10.22

Syntax

```
requestID DirectoryString

controlValue:=SEQUENCE {
{SEQUENCE of requestID}
clientIP String
}
```

Behavior

This control is registered for the following operations:

- Any
- Add
- Bind
- Compare
- Delete
- Extended Operations
- Search
- Modify
- Modrdn

All users, including anonymous, are enabled to send the control. However, there is an environment variable, `SLAPD_AUDIT_DISABLE_NON_ADMIN`, that, when set, restricts the control to the following:

- Local Administrators
- Local Administration Group members
- Global Administration Group members

If `SLAPD_AUDIT_DISABLE_NON_ADMIN` is set to `TRUE`, only audit controls sent by administrators are audited. By default the server enables any user to send this control.

Note: If non-admin users are disabled, and the control is sent by a non-admin, the control is ignored, even if it is critical. If there is additional information required for the control, the error is ignored, and the information is audited

The Administration Daemon honors the control, but audits only one of these controls per operation. The behavior for the Administration Daemon is the same.

Scope The control lasts for the term of one operation. Each operation treats the control the same. If the operation is audited, the additional information sent in the control is audited as well.

Auditing

When the server receives this control the audit plug-in will add the following lines to the audit entry:

```
controlType: <control ID>
criticality: <true | false>
requestID: <request ID sent in the control>
requestID: <request ID sent in the control>
requestID: <request ID sent in the control>
clientIP: <client IP sent in the control>
```

Do not replicate control

Description

This control can be specified for an update operation. When present, a server will not replicate the update to any consumers.

OID 1.3.18.0.2.10.23

Syntax

This control has no value.

Behavior

This control is registered for the following operations:

Add When the control is detected in an add operation, the replication threads in a supplier will not replicate the add operation to the consumer.

Delete When the control is detected in a delete operation, the replication threads in a supplier will not replicate the delete operation to the consumer.

Modify

When the control is detected in a modify operation, the replication threads in a supplier will not replicate the modify operation to the consumer.

Modrdrn

When the control is detected in a modrdrn operation, the replication threads in a supplier will not replicate the modify operation to the consumer.

Any kind of administrators and the Master Server DN are able to send the control.

The Administration Daemon does not support this control.

Scope The control lasts for the term of one operation.

Group authorization control

Description

The Group Authorization Control enables a bound user to assert group membership. The server uses this set of groups in the evaluation of ACLs

for the operation. The control was introduced as a tool for the proxy server. However, this control can be sent by any client.

Note: This control is always enabled.

OID 1.3.18.0.2.10.21

Syntax

```
Group ::= SEQUENCE { groupName LDAPString }
RequestValue ::= SEQUENCE {
    normalized INTEGER { unnormalized(0), normalized(1) };
    Sequence of Group
}
```

The criticality must be set to TRUE in order to protect clients from submitting a request with an unauthorized identity.

Behavior

This control can operate independent of other controls. However, it is often sent with the Proxy Authorization Control. This control is registered for the following operations:

- Any
- Add
- Bind
- Compare
- Delete
- Extended Operations
- Search
- Modify
- Modrdn

The following are enabled to send the control:

- Local Administrators
- Proxy Authorization Group members
- Local Administration Group members
- Global Administration Group members

Only the local admin and local admin group members can assert group membership into the global admin group. Proxy group members and global admin group members do not have the authority to assert group membership into the global admin group.

Note: If the control is sent by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

If there is additional information required for the control, and there is an error in the formatting of that information, the following error returns might occur:

- Missing information – LDAP_OPERATIONS_ERROR
- Additional information – LDAP_OPERATIONS_ERROR
- Invalid information – LDAP_OPERATIONS_ERROR

This control has the following possible return codes:

- LDAP_INSUFFICIENT_ACCESS
- LDAP_OPERATIONS_ERROR

This control is not supported by the Administration Daemon.

Scope The control lasts for the term of one operation. Each operation treats the control the same. The operation is performed assuming that the user is a member of the stated groups. This applies to all back-end servers.

Auditing

This control has a special flag to indicate whether additional information must be audited. If the audit flag `ibm-auditGroupsOnGroupControl` is set to `FALSE`, then the control OID and criticality only are audited. If `ibm-auditGroupsOnGroupControl` is `TRUE`, then the following additional information is audited:

```
controlType: <control ID>
criticality: {true | false}
Normalized: {true | false}
Group: <group sent in request>
Group: <group sent in request>
Group: <group sent in request>
```

Manage DSAIT control

Description

Causes entries with the "ref" attribute to be treated as normal entries, allowing clients to read and modify these entries.

OID 2.16.840.1.113730.3.4.2

Syntax

This control has no value.

Behavior

This control is registered for any operation.

All users are enabled to send the control.

Scope The control lasts for one operation.

Modify groups only control

Description

This control can be used with a `delete`, `modrtn`, or `moddn` operation to cause the server to modify the groups in which it is in a member without deleting or modifying the entry itself. The entry named in the `delete`, `modrtn`, or `moddn` request does not need to exist on the server.

Note: This control is always enabled.

OID 1.3.18.0.2.10.25

Syntax

This control has no value.

Behavior

This control is registered for the following operations:

- Delete
- Modrtn

The following are enabled to send the control:

- Local Administrators
- Local Administration Group members
- Global Administration Group members

Note: If the control is sent by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This control has the following possible return codes:

- LDAP_SUCCESS
- LDAP_DECODING_ERROR
- LDAP_UNWILLING_TO_PERFORM

The Administration Daemon does not support this control.

Scope The control lasts for the term of one operation. The control is only honored when a delete, moddn, or modrdn request goes to the RDBM backend.

Auditing

When the server receives this control the audit plug-in will add the following lines to the audit entry:

```
controlType: <control ID>  
criticality: <true | false>
```

No replication conflict resolution control

Description

When present, a replica server accepts a replicated entry without trying to resolve any replication conflict for this entry. This control can be used by the replication topology extended operation to ensure data consistency between a supplier and a consumer.

Note: If environment variable IBMSLDAPD_REPL_NO_CONFLICT_RESOLUTION is set on a replica, a replica server acts as if all the update requests coming from the suppliers are specified with this control. The replica accepts the replicated entries without attempting to resolve any replication conflicts. This environment variable is useful in a network topology in which one supplier and one or multiple consumers are defined.

OID 1.3.18.0.2.10.27

Syntax

This control has no value.

Behavior

This control is registered for the following operations:

- Add
- Delete
- Modify
- Modrdn

Add Upon receiving such a control in a replicated Add request, a replica server will not try to resolve any replication conflict for this update but accept it and apply it to the replica.

Modify

Upon receiving such a control in a replicated Modify request, a replica server will not try to resolve any replication conflict for this update, but accept it and apply it to the replica.

Only the Master Server DN is able to send the control.

Note: If the control is sent by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

The Administration Daemon does not support this control.

Scope The control lasts for the term of one operation.

Omit group referential integrity control

Description

This control enables an administrator to request that group referential integrity not be performed. The control only applies to modrdn and delete operations. When present on a delete or rename operation, the entry is deleted from or renamed in the directory, but the entry's membership is not removed or renamed in the groups in which the entry is a member.

Note: This control is always enabled.

OID 1.3.18.0.2.10.26

Syntax

This control has no value.

Behavior

This control is registered for the following operations:

- Delete
- Modrdn

The following are enabled to send the control:

- Local Administrators
- Local Administration Group members
- Global Administration Group members

Note: If the control is sent by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This control has the following possible return codes:

- LDAP_SUCCESS
- LDAP_DECODING_ERROR
- LDAP_UNWILLING_TO_PERFORM

The Administration Daemon does not honor the control.

Scope The control lasts for the term of one operation. The control is only honored when a delete, moddn, or modrdn request goes to the RDBM backend.

Auditing

When the server receives this control the audit plug-in will add the following lines to the audit entry:

```
controlType: <control ID>  
criticality: <true | false>
```

Paged search results control

Description

The paged results control is enabled on a search operation and enables a client to request a subset of entries. Subsequent search requests using this control continue to result in the next page of results until the operation is canceled or the last result is returned.

Note: This control can be disabled by setting the Paged Result Limit to 0.

There is also a configuration option which enables an administrator to grant or deny the use of this control to non-administrators (administrators in this case refers to the local admin, local admin group members, and global admin group members). If the `ibm-slapdPagedResAllowNonAdmin` attribute in the `cn=RDBM Backends, cn=IBM Directory, cn=Schemas, cn=Configuration` entry is set to `TRUE`, all users can send paged search requests. If set to `FALSE`, only administrators can send paged search requests.

OID 1.2.840.113556.1.4.319

Syntax

```
realSearchControlValue ::= SEQUENCE {  
    Size INTEGER(0..maxInt),  
    -- requested page size from client  
    -- result set size estimate from server  
    Cookie OCTET STRING }
```

Behavior

This control is registered for the following operations:

- Search

In a default user installation, any user can send this control. If the `ibm-slapdSortSrchAllowNonAdmin` is set to `FALSE`, the use of this control is restricted to administrative users:

- Local Administrators
- Local Administration Group members
- Global Administration Group members

Note: If the control is sent by a user who does not have access, `LDAP_INSUFFICIENT_ACCESS` is returned.

If there is additional information required for the control, and there is an error in the formatting of that information, the following error returns might occur:

- Missing information – `LDAP_DECODING_ERROR`
- Additional information – `LDAP_DECODING_ERROR`
- Invalid information – `LDAP_DECODING_ERROR`

This control has the following possible return codes:

- `LDAP_SUCCESS`
- `LDAP_DECODING_ERROR`
- `LDAP_OPERATIONS_ERROR`
- `LDAP_INSUFFICIENT_ACCESS`
- `LDAP_OTHER`

The Administration Daemon does not support this control.

Scope The control lasts for the term of one operation. The control changes the behavior of a search operation that goes against the RDBM backend. The control requests that the server return the entries in a sorted order. The configuration back-end and schema back-ends do not support this control.

Auditing

When the server receives this control the audit plug-in will add the following lines to the audit entry:

controlType: <control ID>
criticality: <true | false>

Password policy request control

Description

This control is sent by the client application with the requested operation. This control indicates to the server that this client understands Password Policy return values. If the client sends the Password policy request control with the request, the server can send the Password policy request control with the response. The Password policy request control contains extra information about why an operation failed due to a Password Policy problem such as if a client bind request failed because the user's account is locked out. This information is sent to the client on the response in the Password Policy Response Control's value field.

Note: If the Password Policy is disabled, then the Password policy request control is ignored, so no Password policy request control is sent with the response.

Request

OID 1.3.6.1.4.1.42.2.27.8.5.1

Syntax

There is no request value for the control.

Response

OID 1.3.6.1.4.1.42.2.27.8.5.1

Syntax

```
SEQUENCE {  
    warning [0] CHOICE OPTIONAL {  
        timeBeforeExpiration [0] INTEGER (0 .. MaxInt),  
        graceLoginsRemaining [1] INTEGER (0 .. maxInt) }  
    error [1] ENUMERATED OPTIONAL {  
        passwordExpired (0),  
        accountLocked (1),  
        changeAfterReset (2),  
        passwordModNotAllowed (3),  
        mustSupplyOldPassword (4),  
        invalidPasswordSyntax (5),  
        passwordTooShort (6),  
        passwordTooYoung (7),  
        passwordInHistory (8) } }
```

Behavior

This control is registered for the following operations:

- Any
- Add
- Bind
- Compare
- Delete
- Extended Operations
- Search
- Modify
- Modrtn

All users are enabled to send the control. This control has the following possible return codes:

- LDAP_INSUFFICIENT_ACCESS
- LDAP_INVALID_CREDENTIALS
- LDAP_CONSTRAINT_VIOLATION
- LDAP_UNWILLING_TO_PERFORM

The Administration Daemon supports this control. The Administration Daemon checks for this control on the bind operation, and returns the Password policy response control and values if needed. If the Root Administrator has too many bad binds in a row, the Administration Daemon locks out the account and sends the Password Policy response that the account is locked.

Scope The control lasts for the term of one operation. This control indicates to the server that the client application has knowledge of Password Policy and so the server sends a Password policy response control with its response. Along with this response control, there can be a response control value which contains the Password Policy error or warning code and message if one is needed. The other back-ends have no knowledge of this control and so it is ignored.

Proxy authorization control

Description

The Proxy Authorization Control enables a bound user to assert another user's identity. The server uses this asserted identity in the evaluation of ACLs for the operation.

Note: This extended operation is always enabled.

OID 2.16.840.1.113730.3.4.18

Syntax

User DN can be one of the following:

```
dn: <dn value>
<dn value>
RequestValue:: = User DN
```

Behavior

This control can operate independent of other controls. However, it is often sent with the Proxy Authorization Control. This control is registered for the following operations:

- Add
- Bind
- Compare
- Delete
- Extended Operations
- Search
- Modify
- Modrdn

The following are enabled to send the control:

- Local Administrators
- Proxy Authorization Group members
- Local Administration Group members
- Global Administration Group members

No user can assert the identity of the local admin, or local admin group members. Only the local admin and local admin group members can assert the identity of a global admin group member. Global admin group members and proxy group members cannot assert the identity of a global admin group member.

Note: If the control is sent by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

If there is additional information required for the control, and there is an error in the formatting of that information, the following error returns might occur:

- Missing information – LDAP_OPERATIONS_ERROR
- Additional information – LDAP_OPERATIONS_ERROR
- Invalid information – LDAP_OPERATIONS_ERROR

This control has the following possible return codes:

- LDAP_SUCCESS
- LDAP_OPERATIONS_ERROR
- LDAP_INSUFFICIENT_ACCESS
- LDAP_UNWILLING_TO_PERFORM
- LDAP_OTHER
- LDAP_NO_MEMORY
- LDAP_OPERATIONS_ERROR
- LDAP_PARAM_ERROR

This control is not supported by the Administration Daemon.

Scope The control lasts for the term of one operation. Each operation treats the control the same. The operation is performed assuming the asserted user's identity. The control is honored on all operations.

Auditing

When the server receives this control the audit plug-in will add the following lines to the audit entry:

ProxyDN: <proxy dn>

Refresh entry control

Description

This control is returned to a supplier when a consumer server detects a replication conflict during a replicated modify operation. Upon receiving such a control along with an LDAP_OTHER return code, the supplier will retrieve its copy of the entry and send the entry again to the consumer using an add operation to refresh the consumer's version of the entry.

OID 1.3.18.0.2.10.24

Syntax

This control has no value.

Behavior

This control is registered for the following operations:

- Modify

This control is sent in an LDAP response protocol after a conflict is detected on a replicated entry on a consumer. The consumer does not have to specifically bind to the supplier to return such a control. The supplier

has already bound to the consumer. If anybody sends such a control in an LDAP request to any server, the control will be ignored and will have no effect on the server.

The Administration Daemon does not support this control.

Scope The control lasts for the term of one operation. This control is used by a consumer to communicate to its supplier when a replication conflict is detected on the consumer. Once the supplier gets the control along with an LDAP_OTHER return code, the supplier sends the entry again with an intention of bringing the consumer back in sync.

Replication supplier bind control

Description

Gateway servers only send the changes they receive from a gateway to their local servers (servers that reside in the same site as the gateway server, including peer, forwarder or pelican server). They do not send these changes to the other gateway servers. The Replication supplier bind control helps a gateway server to decide which servers to send to and what to send them. When a gateway server binds to its consumers, it sends the control with its serverID as the control value. When a gateway server receives such a control in a bind request, it knows that a gateway server is bound as a supplier.

OID 1.3.18.0.2.10.18

Syntax

```
controlValue :: SEQUENCE {  
    SupplierServerId LDAPString  
}
```

Behavior

This control is registered for the following operations:

- Bind

Only the Master DN is enabled to send this control.

Note: If the control is sent by a user who does not have access, LDAP_UNWILLING_TO_PERFORM is returned.

If there is additional information required for the control, and there is an error in the formatting of that information, the following error returns might occur:

- Missing information – LDAP_OPERATIONS_ERROR
- Additional information – ignored
- Invalid information – ignored

This control has the following possible return codes:

- LDAP_PROTOCOL_ERROR
- LDAP_UNWILLING_TO_PERFORM

The Administration Daemon does not support this control.

Scope The control lasts for the life of the bind session. When the control is received, a server knows that a gateway server is bound as a supplier. Depending on the supplier information, the server can decide to which consumers an entry is to be replicated.

Replication update ID control

Description

This control was created for serviceability. If the supplier server is set to issue the control, each replicated update is accompanied by this control. The data in this control can be used to identify problems with multi-threaded replication and replication conflict resolution. By default, no supplier includes this control.

Note: This control is always enabled.

OID 1.3.18.0.2.10.29

Syntax

<replication agreement DN>:<replication change ID>

These values are set by the supplier.

Behavior

This control is not registered by any operations.

All users are enabled to send the control.

The Administration Daemon does not support this control.

Scope The control lasts for one operation.

Auditing

When the server receives this control the audit plug-in will add the following lines to the audit entry:

```
controlType: OID
criticality: false
value: Replication agreement DN:change ID
```

Server administration control

Description

Allows an update operation by the administrator under conditions when the operation is normally refused (for example, the server is quiesced, the server is a read-only replica, and so forth).

This control can be specified on an update operation (add, modify, modRdn, modDn, delete) by a client bound as an administrator. When present, a server that would normally refuse updates (quiesced server, forwarder or replica), allows the update. The updates are replicated like other updates.

Note: This control needs to be used with user's discretion. With the control, entry updates are allowed under unusual circumstances. Therefore, it is the user's responsibility to ensure the server being updated ends up in a state consistent with the other servers, for example, the timestamp of an entry which is used as the base for replication conflict resolution in IBM Tivoli Directory Server 6.0 might be different on different servers if the entry gets updated individually on those servers with this control.

OID 1.3.18.0.2.10.15

Syntax

This control has no value.

Behavior

This control is registered for the following operations:

- Add
- Delete
- Modify
- Modrdn

The following are enabled to send the control:

- Local Administrators
- Local Administration Group Member
- Global Administration Group Member

Note: If the control is sent by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

The Administration Daemon does not support this control.

Scope The control lasts for one operation. When the control is received, a server knows that a gateway server is bound as a supplier. Depending on the supplier information, the server can decide to which consumers an entry is to be replicated.

Sorted search results control

Description

The sorted search results control enables a client to receive search results sorted by a sort key.

Note: This control can be disabled by setting the `ibm-slapdSortKeyLimit` to 0.

There is also a configuration option which enables an administrator to grant or deny the use of this control to non-administrators (administrators in this case refers to the local admin, local admin group members, and global admin group members). If the `ibm-slapdSortSrchAllowNonAdmin` attribute in the `cn=RDBM Backends`, `cn=IBM Directory`, `cn=Schemas`, `cn=Configuration` entry is set to TRUE, then all users are enabled to use the sorted search. If set to FALSE, only administrators can use the sorted search.

OID 1.2.840.113556.1.4.473

Syntax

The `controlValue` is an OCTET STRING whose value is the BER encoding of a value with the following SEQUENCE:

```
SortKeyList ::= SEQUENCE of SEQUENCE {
    AttributeType AttributeDescription,
    OrderingRule [0] MatchingRuleId OPTIONAL,
    ReverseOrder [1] BOOLEAN DEFAULT FALSE }
```

Behavior

This control is registered for the following operations:

- Search

In a default user installation, any user can send this control. If the `ibm-slapdSortSrchAllowNonAdmin` is set to FALSE, the use of this control is restricted to administrative users:

- Local Administrators
- Local Administration Group members
- Global Administration Group members

Note: If the control is sent by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

If there is additional information required for the control, and there is an error in the formatting of that information, the following error returns might occur:

- Missing information – LDAP_DECODING_ERROR
- Additional information – LDAP_DECODING_ERROR
- Invalid information – LDAP_DECODING_ERROR

This control has the following possible return codes:

- LDAP_SUCCESS
- LDAP_DECODING_ERROR
- LDAP_OPERATIONS_ERROR
- LDAP_INSUFFICIENT_ACCESS
- LDAP_OTHER

The Administration Daemon does not support this control.

Scope The control lasts for the term of one operation. The control changes the behavior of a search operation that goes against the RDBM backend. The control requests that the server return the entries in a sorted order. The configuration back-end and schema back-ends do not support this control.

Auditing

When the server receives this control the audit plug-in will add the following lines to the audit entry:

```
controlType: <control ID>
criticality: <true | false>
```

Subtree delete control

Description

This control is attached to a delete request. This control indicates that the specified entry and all descendent entries are to be deleted. However, if the subtree is an active replication context, the control does not take effect and an LDAP_UNWILLING_TO_PERFORM message is returned. This means if the subtree to be deleted contains any replication agreements that the server uses to replicate, then the subtree cannot be deleted using this control.

OID 1.2.840.113556.1.4.805

Syntax

This control has no value.

Behavior

This control is registered for the following operations:

- Delete

The following are enabled to send the control:

- Local Administrators
- Local Administration Group members
- Global Administration Group members

- Master server DN

Note: If the control is sent by a user who does not have access, LDAP_INSUFFICIENT_ACCESS is returned.

This control has the following possible return codes:

- LDAP_INSUFFICIENT_ACCESS
- LDAP_UNWILLING_TO_PERFORM

The Administration Daemon does not support this control.

Scope The control lasts for the term of one delete operation. The delete operation not only deletes the base entry specified in the request, but also deletes all the descendent entries.

Transaction control

Description

The Transaction control is sent along with update operations performed within a transaction.

Note: This control is enabled by default, but can be disabled by changing the value in the configuration file for the `ibm-slapdTransactionEnable` attribute.

The `ibm-slapdTransactionEnabled` attribute is in the configuration file in the `cn=Transaction,cn=configuration` entry. If the value is set to `FALSE`, transactions are not enabled. If set to `TRUE`, transactions are enabled. Transactions can also be enabled or disabled using the Web Administration tool.

OID 1.3.18.0.2.10.5

Syntax

The `controlValue` is set to the transaction ID returned in the `StartTransaction` response.

Behavior

This control is registered for the following operations:

- Add
- Delete
- Modify
- Modrdn

Any user can send this control.

If there is additional information required for the control, and the transaction ID sent in the control does not match the transaction ID on the connection, then `LDAP_PROTOCOL_ERROR` is returned.

This control has the following possible return codes:

- LDAP_SUCCESS
- LDAP_PROTOCOL_ERROR
- LDAP_TIMELIMIT_EXCEEDED
- LDAP_SIZELIMIT_EXCEEDED

The Administration Daemon does not support this control.

Scope The control lasts for the term of one operation, but must be sent only in a transactional context. When the control is sent only with an update operation to the RDBM backend, the server holds the update until an end-transaction request is received. The control is only supported on updated operations performed in a transactional context (a start transaction extended operation must be performed first).

Auditing

When the server receives this control the audit plug-in will add the following lines to the audit entry:

```
controlType: <control ID>  
criticality: <true | false>
```

Appendix G. Building LDAP-enabled applications

The following table lists the hardware/software requirements needed in order to build your own LDAP-enabled applications (applications with LDAP client capabilities, which enable communication with any LDAP server):

S.No.	Platform	Hardware	Software – Build	Software – Test
1	AIX		5.1 – VAC 6.0.0.5	5.1, 5.2, 5.3
3a	Linux Intel (IA32)		RHAS2.1 – gccv3.2.3	SLES8, RHEL3
4	Linux on i/p Series		SLES8 – gcc3.2	SLES8, RHEL3
5a	Linux on zSeries® (31bit)		Cross-compile on S.No. 5b	SLES8, RHEL3
6	Solaris		Solaris 8	Solaris 8, 9
7a	Windows – 32 bit		Win 2000 – VC++ 6 fp5 Win 2003/XP – MS Platform SDK 2003	Win 2000, 2003, XP (client only)

Appendix H. Client libraries

Both the 32-bit as well as the 64-bit libraries have the same names. The following table lists the libraries being built for IBM Tivoli Directory Server 6.0 as part of client:

Libraries	Operating Systems				
	AIX	HPUX	Linux	Solaris	Windows – IA32
idsldap_ plugin_ ibm_gsskrb	Y	NA	NA	NA	NA
idsldap_ plugin_ sasl_ cram-md5	Y	Y	Y	Y	Y
idsldap_ plugin_ sasl_ digest-md5	Y	Y	Y	Y	Y
libidsldap	Y	Y	Y	Y	Y
libidsldapn	Y	NA	NA	NA	Y
libids ldapstatic	Y	Y	Y	Y	Y
libids ldapstaticn	Y	NA	NA	NA	Y
libids ldapiconv	Y	Y	Y	Y	Y
libidsldif	NA	Y	Y	NA	NA
libids ldifstatic	Y	Y	Y	Y	Y
libibm ldapdbg	Y	Y	Y	Y	Y
ldap	NA	NA	NA	NA	Y
ldapstatic	NA	NA	NA	NA	Y

Note: The dynamic version of libldif is available on Linux, but not on Solaris.

Legend:

Y This library is 64-bit recertified on the corresponding operating system.

NA This library is not 64-bit recertified, or it is not valid for the corresponding operating system.

Hence the architecture (32-bit or 64-bit) used for those binaries is the one that will be used for these libraries, as well. Consequently these libraries will be placed in the appropriate folder (lib or lib64).

Please note that the following library extensions are applicable for each platform:

Platform	Static library	Shared (Dynamic) library
AIX	.a	.a
Linux	.a	.so
Solaris	.a	.so
Windows	.lib	.dll

The following table identifies the library name changes in IBM Tivoli Directory Server 6.0 with regards to IBM Tivoli Directory Server 5.2:

Library Name in IBM Tivoli Directory Server 5.2	Library Name in IBM Tivoli Directory Server 6.0
libldapstatic	libidsldapstatic
libldapstaticn	libidsldapstaticn
libldif (Static Version)	libidsldifstatic

Appendix I. Sample Makefile

In IBM Tivoli Directory Server 6.0, the sample Makefile (makefile.ex) is updated with the rules and information on building 64-bit clients. These updates are in addition to the already existing rules and information on building 32-bit clients. The platforms, on which the 64-bit binaries or objects can be generated, are listed in Appendix G, "Building LDAP-enabled applications," on page 263.

The sample Makefile lists the 64-bit compilers/linkers to be used along with the relevant flags to be passed. It also lists the 64-bit libraries, needed to build the customized LDAP clients.

Note: You must have the prerequisite compat-glibc library. Use the following command to retrieve this library:

```
up2date compat-glibc
```

This library is available on your operating system CD. Without this library, you will get the following errors when compiling:

```
# make -f makefile.ex ldapdelete
mkdir -p 32
gcc -I../include -I/usr/include -DLINUX -D_GCC3 -o 32/ldapdelete
ldapdelete.c -L../lib -L/opt/ibm/ldap/V6.0/lib -lpthread -ldl
-libldapstatic -libldapdbgstatic -lldifstatic -lldapiconvstatic
-lmsgstatic
../lib/libibmldapstatic.a(ldap_open.o)(.text+0x2b7): In
function `lower': /project/ldapdev/build/ldapdevsb/src/libraries
/libldap/ldap_open.c:338: undefined reference to `__ctype_b'
../lib/libibmldapstatic.a(ldap_utils.o)(.text+0x609): In
function `ldap_path_is_found': /project/ldapdev/build/ldapdevsb
/src/libraries/libldap/ldap_utils.c:362: undefined reference
to `__ctype_b'
../lib/libibmldapstatic.a(ldapdns.o)(.text+0x56): In
function `dumpBuf': /project/ldapdev/build/ldapdevsb/src
/libraries/libldap/ldapdns.c:234: undefined reference
to `__ctype_b'
../lib/libibmldapstatic.a(ldapdns.o)(.text+0x478): In
function `readConfName': /project/ldapdev/build/ldapdevsb/src
/libraries/libldap/ldapdns.c:401: undefined reference
to `__ctype_b'
../lib/libibmldapstatic.a(ldapdns.o)(.text+0x4f6):/project
/ldapdev/build/ldapdevsb/src/libraries/libldap/ldapdns.c:411:
undefined reference to `__ctype_b'
../lib/libibmldapstatic.a(ldapdns.o)(.text+0x58c):/project
/ldapdev/build/ldapdevsb/src/libraries/libldap/ldapdns.c:430:
more undefined references to `__ctype_b' follow
collect2: ld returned 1 exit status
make: *** [ldapdelete] Error 1
```

The following is the sample makefile for Solaris:

```
#
# @(#)58 1.1.3.15 src/clients/tools/makefile.sparc_solaris_2, ldap.clients,
ldapdev 2/26/04 03:42:59
#
#-----
#
# COMPONENT_NAME: examples
#
# ABSTRACT: makefile to generate the example LDAP client programs
```

```

#
# ORIGINS: 202,27
#
# (C) COPYRIGHT International Business Machines Corp. 1997, 1998, 2001
# 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) 1994 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.
#-----
#
# This makefile will build the example programs whose source is contained
# in this directory. The four programs generated are:
# ldapsearch
# ldapmodify
# ldapadd (a hard-link to ldapmodify)
# ldapmodrdn
# ldapdelete
# ldapchangepwd
# ldapexop
# In addition to being examples of the use of the LDAP client api, these
# programs are useful command line utilities. See the README file for
# more details.
#
# default definitions for Unix utilities (may be changed here)
CC = CC
RM = rm -f
HARDLN = ln
MKDIR = mkdir -p

# Change this BITS flag to 32/64 depending upon the architecture of the output
# binaries desired. In addition you would need to comment/uncomment the
# appropriate rules under the 32-bit and 64-bit sections

BITS = 32

#####
## General compiler options
#####

DEFINES = -Dsunos5
# Note: replace ../include with the appropriate path to the LDAP header files.
# Also append the include path for the compiler headers (if required).

INCLUDES= -I../include -I/usr/include
CFLAGS= $(INCLUDES) $(DEFINES) -DNEEDPROTOS -mt -w

#####
## Options for building 32-bit targets on Solaris
#####
# Use the following definition to link the sample programs with
# the shared LDAP library. There is no dynamic version of libldif, hence we
# need to see the static version of libldif
#
# CLIENT_LIBS = -lldapiconv -lldifstatic -libmldapdbg -libldap -lmsg

```

```

#
# Or use this definition to link the LDAP library statically:
CLIENT_LIBS = -lldifstatic -libldapstatic -libldapdbgstatic -lldapiconvstatic
-lmsgstatic
#
LIBS = -L../lib -L/opt/IBM/ldap/V6.0/lib -L/usr/lib -lsocket -lpthread -lnsl -ldl
LFLAGS = $(LIBS) $(CLIENT_LIBS)
#####

#####
## Options for building 64-bit targets on Solaris ##
#####
# use the following compiler and linker flags
#
# CFLAGS += -xarch=v9
#
# Use the following definition to link the sample programs with
# the shared LDAP library.
#
# CLIENT_LIBS = -lldapiconv -lldifstatic -libldapdbg -libldap -lmsg
#
# Or use this definition to link the LDAP library statically:
# CLIENT_LIBS = -lldifstatic -libldapstatic -libldapdbgstatic -lldapiconvstatic
-lmsgstatic
#
# LIBS = -L../lib64 -L/opt/IBM/ldap/V6.0/lib64 -L/usr/lib/sparcv9 -lsocket
-lpthread -lm -lnsl -ldl
# LFLAGS = $(LIBS) $(CLIENT_LIBS)
#####

#####
## Targets ##
#####

all: ldapsearch ldapmodify ldapdelete ldapmodrdn ldapadd ldapchangepwd ldapexop

ldapsearch:
$(MKDIR) $(BITS)
$(CC) $(CFLAGS) -o $(BITS)/$@ ldapsearch.c $(LFLAGS)

ldapmodify:
$(MKDIR) $(BITS)
$(CC) $(CFLAGS) -o $(BITS)/$@ ldapmodify.c $(LFLAGS)

ldapdelete:
$(MKDIR) $(BITS)
$(CC) $(CFLAGS) -o $(BITS)/$@ ldapdelete.c $(LFLAGS)

ldapmodrdn:
$(MKDIR) $(BITS)
$(CC) $(CFLAGS) -o $(BITS)/$@ ldapmodrdn.c $(LFLAGS)

ldapchangepwd:
$(MKDIR) $(BITS)
$(CC) $(CFLAGS) -o $(BITS)/$@ ldapchangepwd.c $(LFLAGS)

ldapexop:
$(MKDIR) $(BITS)
$(CC) $(CFLAGS) -o $(BITS)/$@ ldapexop.c $(LFLAGS)

ldapadd: ldapmodify
$(RM) $(BITS)/$@
$(HARDLN) $(BITS)/ldapmodify $(BITS)/ldapadd

clean:

```

```
$(RM) *.o core a.out $(BITS)/*.o $(BITS)/core $(BITS)/a.out $(BITS)/ldapsearch  
$(BITS)ldapmodify $(BITS)/ldapdelete \  
$(BITS)/ldapmodrdrn $(BITS)/ldapadd $(BITS)/ldapchangepwd $(BITS)/ldapexop
```

You can find the sample Makefile (makefile.ex) in `<ldap_home>/examples`.

Appendix J. 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 {
```

```
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[] = "@(#)17 1.35 11/18/02 progref.idd, ldap, 5.1 15:20:20";  
/*  
 * 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

```

```

#ifdef -win32
#ifdef -GCC3
#include <errno.h>
#else
extern int errno;
#endif
#endif

/*Required for password prompting*/
#ifdef -win32
#include <conio.h>
#else
/*termios.h is defined by POSIX*/
#include <termios.h>
#endif

/* Global variables */
static LDAP *ld = NULL; /* LDAP session 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. */
#ifdef _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:";
#ifdef _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

    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 );
}
        }
    }
}
}

```



```

    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_deleteolrdn = 0;
int deleteolrdn = 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 ((strcmp(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 && strcmp( 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_deleteolddrn = 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_deleteolddrn ) {
        if ( strcasecmp( type, T_DELETEOLDRDNSTR ) == 0 ) {
deleteolddrn = ( *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, deleteolddrn );
    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/slappd.repllog 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 succeeded 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\V6.0

#####
# 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\
odbc32.lib wsoc32.lib

```

```

# Use the following definition to link the sample programs statically.
#CLIENT_LIBS = ldapstatic.lib libidsldifstatic.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

```

Appendix K. Support information

This section describes the following options for obtaining support for IBM products:

- “Searching knowledge bases”
- “Obtaining fixes”
- “Contacting IBM Software Support” on page 298

Searching knowledge bases

If you have a problem with your IBM software, you want it resolved quickly. Begin by searching the available knowledge bases to determine whether the resolution to your problem is already documented.

Search the information center on your local system or network

IBM provides extensive documentation that can be installed on your local computer or on an intranet server. You can use the search function of this information center to query conceptual information, instructions for completing tasks, reference information, and support documents.

Search the Internet

If you cannot find an answer to your question in the information center, search the Internet for the latest, most complete information that might help you resolve your problem. To search multiple Internet resources for your product, expand the product folder in the navigation frame to the left and select **Web search**. From this topic, you can search a variety of resources including:

- IBM technotes
- IBM downloads
- IBM Redbooks™
- IBM developerWorks®
- Forums and newsgroups
- Google

Obtaining fixes

A product fix might be available to resolve your problem. You can determine what fixes are available for your IBM software product by checking the product support Web site:

1. Go to the IBM Software Support Web site (<http://www.ibm.com/software/support>).
2. Under **Products A - Z**, select your product name. This opens a product-specific support site.
3. Under **Self help**, follow the link to **All Updates**, where you will find a list of fixes, fix packs, and other service updates for your product. For tips on refining your search, click **Search tips**.
4. Click the name of a fix to read the description and optionally download the fix.

To receive weekly e-mail notifications about fixes and other news about IBM products, follow these steps:

1. From the support page for any IBM product, click **My support** in the upper-right corner of the page.
2. If you have already registered, skip to the next step. If you have not registered, click register in the upper-right corner of the support page to establish your user ID and password.
3. Sign in to **My support**.
4. On the My support page, click **Edit profiles** in the left navigation pane, and scroll to **Select Mail Preferences**. Select a product family and check the appropriate boxes for the type of information you want.
5. Click **Submit**.
6. For e-mail notification for other products, repeat Steps 4 and 5.

For more information about types of fixes, see the *Software Support Handbook* (<http://techsupport.services.ibm.com/guides/handbook.html>).

Contacting IBM Software Support

IBM Software Support provides assistance with product defects.

Before contacting IBM Software Support, your company must have an active IBM software maintenance contract, and you must be authorized to submit problems to IBM. The type of software maintenance contract that you need depends on the type of product you have:

- For IBM distributed software products (including, but not limited to, Tivoli, Lotus®, and Rational® products, as well as DB2® and WebSphere® products that run on Windows or UNIX operating systems), enroll in Passport Advantage® in one of the following ways:
 - **Online:** Go to the Passport Advantage Web page (http://www.lotus.com/services/passport.nsf/WebDocs/Passport_Advantage_Home) and click **How to Enroll**
 - **By phone:** For the phone number to call in your country, go to the IBM Software Support Web site (<http://techsupport.services.ibm.com/guides/contacts.html>) and click the name of your geographic region.
- For IBM eServer™ software products (including, but not limited to, DB2 and WebSphere products that run in zSeries, pSeries™, and iSeries™ environments), you can purchase a software maintenance agreement by working directly with an IBM sales representative or an IBM Business Partner. For more information about support for eServer software products, go to the IBM Technical Support Advantage Web page (<http://www.ibm.com/servers/eserver/techsupport.html>).

If you are not sure what type of software maintenance contract you need, call 1-800-IBMSERV (1-800-426-7378) in the United States or, from other countries, go to the contacts page of the IBM Software Support Handbook on the Web (<http://techsupport.services.ibm.com/guides/contacts.html>) and click the name of your geographic region for phone numbers of people who provide support for your location.

Follow the steps in this topic to contact IBM Software Support:

1. Determine the business impact of your problem.
2. Describe your problem and gather background information.

3. Submit your problem to IBM Software Support.

Determine the business impact of your problem

When you report a problem to IBM, you are asked to supply a severity level. Therefore, you need to understand and assess the business impact of the problem you are reporting. Use the following criteria:

Severity 1	Critical business impact: You are unable to use the program, resulting in a critical impact on operations. This condition requires an immediate solution.
Severity 2	Significant business impact: The program is usable but is severely limited.
Severity 3	Some business impact: The program is usable with less significant features (not critical to operations) unavailable.
Severity 4	Minimal business impact: The problem causes little impact on operations, or a reasonable circumvention to the problem has been implemented.

Describe your problem and gather background information

When explaining a problem to IBM, be as specific as possible. Include all relevant background information so that IBM Software Support specialists can help you solve the problem efficiently. To save time, know the answers to these questions:

- What software versions were you running when the problem occurred?
- Do you have logs, traces, and messages that are related to the problem symptoms? IBM Software Support is likely to ask for this information.
- Can the problem be re-created? If so, what steps led to the failure?
- Have any changes been made to the system? (For example, hardware, operating system, networking software, and so on.)
- Are you currently using a workaround for this problem? If so, please be prepared to explain it when you report the problem.

Submit your problem to IBM Software Support

You can submit your problem in one of two ways:

- **Online:** Go to the "Submit and track problems" page on the IBM Software Support site (<http://www.ibm.com/software/support/probsub.html>). Enter your information into the appropriate problem submission tool.
- **By phone:** For the phone number to call in your country, go to the contacts page of the IBM Software Support Handbook on the Web (techsupport.services.ibm.com/guides/contacts.html) and click the name of your geographic region.

If the problem you submit is for a software defect or for missing or inaccurate documentation, IBM Software Support creates an Authorized Program Analysis Report (APAR). The APAR describes the problem in detail. Whenever possible, IBM Software Support provides a workaround for you to implement until the APAR is resolved and a fix is delivered. IBM publishes resolved APARs on the IBM product support Web pages daily, so that other users who experience the same problem can benefit from the same resolutions.

For more information about problem resolution, see Searching knowledge bases and Obtaining fixes.

Appendix L. 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 MU5A46
11301 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.

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

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

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

AIX	Lotus	SecureWay
DB2	Passport Advantage	Tivoli
developerWorks	pSeries	WebSphere
eServer	RACF	World Registry
IBM	Rational	z/OS
ibm.com	Redbooks	zSeries
iSeries		

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.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group.

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

Index

A

- Account status extended operation 213
- AES bind control 246
- API
 - categories 57
 - deprecated 209
 - plug-ins 188
 - usage 2
- applications
 - LDAP-enabled 263
- Attribute type extended operations 214
- attributes
 - ldap 90
- Audit control 247

B

- Begin transaction extended operation 216
- binding
 - sasl 61
 - secure 61
 - simple 61

C

- Cascading replication operation extended operation 217
- certificate authority 171
 - distinguished names 177
 - receiving a certificate 173
- certificate requests 175
- certificates 171
 - Certificate Authority 171
 - receiving a certificate 173
- Clear log extended operation 242
- client controls 78
- client libraries 265
- client utilities
 - idsldapadd 5
 - idsldapchangepwd 11
 - idsldapdelete 14
 - idsldapdiff 18
 - idsldapexop 27
 - idsldapmodify 5
 - idsldapmodrdrn 37
 - idsldapsearch 41
 - ldapadd 5
 - ldapchangepwd 11
 - ldapdelete 14
 - ldapdiff 18
 - ldapexop 27
 - ldapmodify 5
 - ldapmodrdrn 37
 - ldapsearch 41
- code page
 - getting 68
 - setting 68
 - translating 68

- commands
 - ibmdirctl 54
 - idsldapadd 5
 - idsldapdelete 14
 - idsldapdiff 18
 - idsldapexop 27
 - idsldapmodify 5
 - idsldapmodrdrn 37
 - idsldapsearch 41
 - idsldaptrace 50
 - ldapadd 5
 - ldapdelete 14
 - ldapdiff 18
 - ldapexop 27
 - ldapmodify 5
 - ldapmodrdrn 37
 - ldapsearch 41
 - ldaptrace 50
- compare operations 76
- Control queue extended operation 220
- Control replication extended operation 218
- controls 95
 - ldap 78
 - OIDs 211, 244
- counting
 - entries 92
 - references 92
- counting values 98
- customer support
 - see Software Support 298

D

- data interchange format 203
- deleting
 - keys 174
- deleting entries 14, 80
- Directory C-Client SDK overview 1
- directory operations 1
- distinguished name 201
 - formal definition 202
 - informal definition 201
- DN normalization extended operation 221
- DNs 201
- DNS 135
- DNS configuration file
 - examples 144
- Do not replicate control 248
- dynamic schema 197
 - changes 199
- Dynamic server trace extended operation 222
- Dynamic update requests extended operation 223

E

- End transaction extended operation 224

- entry
 - adding 59
 - changing passwords 11
 - counting 92
 - deleting 14, 80
 - idsldapchangepwd 11
 - modifying 37
 - referencubg 92
 - searching 41
- error codes 193
- error numbers 81, 83
- errors
 - ldap 81, 83
- event notification 181
 - example 182
 - registration request 181
 - registration response 181
 - unregistering 182
- Event notification register request extended operation 225
- Event notification unregister request extended operation 226
- example
 - event notification 182
 - LDIF 203
 - Version 1 205
 - limited transaction support 272
- examples
 - DNS configuration file 144
- exporting
 - keys 176
- extended operations 27, 88
- OIDs 211

F

- fixes, obtaining 297
- freeing storage
 - BER 111
 - controls 111
 - memory 111
 - messages 111

G

- Get lines extended operation 243
- Get number of lines extended operation 244
- getting values 98
- global security 171
- Group authorization control 248
- Group evaluation extended operation 227
- GSKit 171

H

- handling routines 95

I

- IANA character sets 206
- IBM Tivoli Directory Server 6.0
 - Kerberos 3
 - updates 3
- ibmdirctl 54
- iconv 68
- idsldapadd 5
- idsldapchangepwd 11
- idsldapdelete 14
- idsldapdiff 18
- idsldapexop 27
- idsldapmodify 5
- idsldapmodrdn 37
- idsldapsearch 41
- idsldaptrace 50
- importing
 - keys 177
- information centers, searching to find
 - software problem resolution 297
- initializing libraries 100
- Internet, searching to find software
 - problem resolution 297

K

- key
 - certificate request for existing
 - key 178
 - changing the database password 173
 - defaults 175
 - deleting 174
 - exporting 176
 - importing 177
 - private 171
 - public 171
 - self-signing 175
 - showing information about 174
 - trusted root removal 178
 - trusted roots 177
- key pairs 171
- keyring file
 - migration 179
- Kill connection extended operation 228
- knowledge bases, searching to find
 - software problem resolution 297

L

- language support 206
- LDAP
 - API overview 1
 - utilities 5
 - version support 1
- ldap attributes 90
- ldap controls
 - client 78
 - server 78
- LDAP SSL function codes 193
- LDAP trace facility extended
 - operation 229
- ldap_abandon 57
- ldap_add 59
- LDAP-enabled applications 263
- ldapadd 5
- ldapchangepwd 11

- ldapdelete 14
- ldapdiff 18
- ldapexop 27
- ldapmodify 5
- ldapmodrdn 37
- ldapsearch 41
- ldaptrace 50
- LDIF 203
- leaving an operation 57
- libraries
 - client 265
 - initialization 100
- limited transactions 271
- Log access extended operations 241

M

- makefile
 - sample 267
- Manage DSAIT control 250
- memory
 - freeing 111
- messages
 - ldap 113
- migration
 - keyring file 179
- Modify groups only control 250
- modify operations 114
- modifying entries 37

N

- No replication conflict resolution
 - control 251
- notification
 - event 181

O

- OIDs
 - AES bind control 246
 - Audit control 247
 - controls 211, 244
 - Do not replicate control 248
 - extended operations 211
 - Group authorization control 248
 - Manage DSAIT control 250
 - Modify groups only control 250
 - No replication conflict resolution control 251
 - Omit group referential integrity control 252
 - Paged search results control 252
 - Password policy request control 254
 - Proxy authorization control 255
 - Refresh entry control 256
 - Replication supplier bind control 257
 - Replication update ID control 258
 - Server administration control 258
 - Sorted search results control 259
 - Subtree delete control 260
 - Transaction control 261

OIDs

- Account status extended
 - operation 213

OIDs (continued)

- Attribute type extended
 - operations 214
- Begin transaction extended
 - operation 216
- Cascading replication operation
 - extended operation 217
- Clear log extended operation 242
- Control queue extended
 - operation 220
- Control replication extended
 - operation 218
- DN normalization extended
 - operation 221
- Dynamic server trace extended
 - operation 222
- Dynamic update requests extended
 - operation 223
- End transaction extended
 - operation 224
- Event notification register request
 - extended operation 225
- Event notification unregister request
 - extended operation 226
- Get lines extended operation 243
- Get number of lines extended
 - operation 244
- Group evaluation extended
 - operation 227
- Kill connection extended
 - operation 228
- LDAP trace facility extended
 - operation 229
- Log access extended operations 241
- Quiesce or unquiesce replication
 - context extended operation 230
- Replication error log extended
 - operation 232
- Replication topology extended
 - operation 233
- Start TLS extended operation 235
- Start, stop server extended
 - operations 234
- Unique attributes extended
 - operation 236
- Update configuration extended
 - operation 237
- Update event notification extended
 - operation 239
- Update log access extended
 - operation 239
- User type extended operation 240

- Omit group referential integrity
 - control 252
- operations
 - comparing 76
 - directory-related 1
 - extended 27, 88
 - renaming 127
 - results 129
 - searching 131

P

- paged results 117
- Paged search results control 252
- parsing 120

- password policy 123
- Password policy request control 254
- passwords
 - changing 11
- pblock 185
- plug-ins
 - APIs 188
 - initializing 186
 - registration 124
 - restrictions 186
 - sample SASL plug-in 190
 - SASL 185
 - writing your own SASL plug-in 188
- problem determination
 - describing problem for IBM Software Support 299
 - determining business impact for IBM Software Support 299
 - submitting problem to IBM Software Support 299
- proxy authorization 74
- Proxy authorization control 255

Q

- Quiesce or unquiesce replication context extended operation 230

R

- rdn 37
- rebinding 61
- records
 - SRV 148
 - TXT 148
- reference
 - entry 92
- Refresh entry control 256
- registration
 - plug-ins 124
- rename operations 127
- Replication error log extended operation 232
- Replication supplier bind control 257
- Replication topology extended operation 233
- Replication update ID control 258
- results 129
 - displaying 3
- routines
 - handling 95

S

- sample makefile 267
- schema
 - changes 199
 - dynamic 197
 - queries 197
- searching 131
- searching entries 41
- secure connections 152
- secure socket layer 3
- security 171
- self-signing keys 175
- Server administration control 258

- server controls 78
- server information
 - DNS 135
- server utilities
 - ibmdirctl 54
 - idsldaptrace 50
 - ldaptrace 50
- Software Support
 - contacting 298
 - describing problem for IBM Software Support 299
 - determining business impact for IBM Software Support 299
 - submitting problem to IBM Software Support 299
- sorted search 45
 - idsldapsearch 45
- Sorted Search and Paged Results
 - Server side sorting of search results 167
 - Simple paged results of search results 119
- Sorted search results control 259
- SRV records 148
- SSL 3, 55
 - cipher support 152
 - starting 152
- SSL environment 164
- ssl_environment_init 164
- Start TLS extended operation 235
- Start, stop server extended operations 234
- storage
 - freeing 111
- subtree
 - comparing 18
- Subtree delete control 260

T

- TLS 55, 159, 160
- Transaction control 261
- transactions
 - limited support 271
- translating locales 68
- trusted root 177
- trusted roots 155
- TXT records 148

U

- unbinding 61
- Unique attributes extended operation 236
- Update configuration extended operation 237
- Update event notification extended operation 239
- Update log access extended operation 239
- URLs 3, 161
- User type extended operation 240
- UTF-8 68, 206
- utilities
 - idsldapadd 5
 - idsldapchangePwd 11

- utilities (*continued*)
 - idsldapdelete 14
 - idsldapdiff 18
 - idsldapexop 27
 - idsldapmodify 5
 - idsldapmodrdn 37
 - idsldapsearch 41
 - LDAP 5
 - ldapadd 5
 - ldapchangePwd 11
 - ldapdelete 14
 - ldapdiff 18
 - ldapexop 27
 - ldapmodify 5
 - ldapmodrdn 37
 - ldapsearch 41

V

- values
 - counting 98
 - getting 98
- version 3 1
- version support 1



Printed in USA

SC32-1675-00

