

LDAP SDK for C Programming Guide

Sun™ Java System Directory Server Resource Kit

Version 5.2.1

817-6707-05
June 2004

Copyright © 2004 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at <http://www.sun.com/patents> and one or more additional patents or pending patent applications in the U.S. and in other countries.

THIS PRODUCT CONTAINS CONFIDENTIAL INFORMATION AND TRADE SECRETS OF SUN MICROSYSTEMS, INC. USE, DISCLOSURE OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR EXPRESS WRITTEN PERMISSION OF SUN MICROSYSTEMS, INC.

U.S. Government Rights - Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

This distribution may include materials developed by third parties.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and in other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, Java, Solaris, JDK, Java Naming and Directory Interface, JavaMail, JavaHelp, J2SE, iPlanet, the Duke logo, the Java Coffee Cup logo, the Solaris logo, the SunTone Certified logo and the Sun ONE logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon architecture developed by Sun Microsystems, Inc.

Legato and the Legato logo are registered trademarks, and Legato NetWorker, are trademarks or registered trademarks of Legato Systems, Inc. The Netscape Communications Corp logo is a trademark or registered trademark of Netscape Communications Corporation.

The OPEN LOOK and Sun(TM) Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

Products covered by and information contained in this service manual are controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright © 2004 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, Etats-Unis. Tous droits réservés.

Sun Microsystems, Inc. détient les droits de propriété intellectuels relatifs à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et ce sans limitation, ces droits de propriété intellectuelle peuvent inclure un ou plus des brevets américains listés à l'adresse <http://www.sun.com/patents> et un ou les brevets supplémentaires ou les applications de brevet en attente aux Etats - Unis et dans les autres pays.

CE PRODUIT CONTIENT DES INFORMATIONS CONFIDENTIELLES ET DES SECRETS COMMERCIAUX DE SUN MICROSYSTEMS, INC. SON UTILISATION, SA DIVULGATION ET SA REPRODUCTION SONT INTERDITES SANS L'AUTORISATION EXPRESSE, ECRITE ET PREALABLE DE SUN MICROSYSTEMS, INC.

Cette distribution peut comprendre des composants développés par des tierces parties.

Des parties de ce produit pourront être dérivées des systèmes Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, Java, Solaris, JDK, Java Naming and Directory Interface, JavaMail, JavaHelp, J2SE, iPlanet, le logo Duke, le logo Java Coffee Cup, le logo Solaris, le logo SunTone Certified et le logo Sun[tm] ONE sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays.

Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

Legato, le logo Legato, et Legato NetWorker sont des marques de fabrique ou des marques déposées de Legato Systems, Inc. Le logo Netscape Communications Corp est une marque de fabrique ou une marque déposée de Netscape Communications Corporation.

L'interface d'utilisation graphique OPEN LOOK et Sun(TM) a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui, en outre, se conforment aux licences écrites de Sun.

Les produits qui font l'objet de ce manuel d'entretien et les informations qu'il contient sont régis par la législation américaine en matière de contrôle des exportations et peuvent être soumis au droit d'autres pays dans le domaine des exportations et importations. Les utilisations finales, ou utilisateurs finaux, pour des armes nucléaires, des missiles, des armes biologiques et chimiques ou du nucléaire maritime, directement ou indirectement, sont strictement interdites. Les exportations ou réexportations vers des pays sous embargo des Etats-Unis, ou vers des entités figurant sur les listes d'exclusion d'exportation américaines, y compris, mais de manière non exclusive, la liste de personnes qui font objet d'un ordre de ne pas participer, d'une façon directe ou indirecte, aux exportations des produits ou des services qui sont régis par la législation américaine en matière de contrôle des exportations et la liste de ressortissants spécifiquement désignés, sont rigoureusement interdites.

LA DOCUMENTATION EST FOURNIE "EN L'ETAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFACON.

Contents

List of Figures	13
List of Tables	15
List of Code Examples	23
Preface	31
Audience for This Guide	31
Directory Server Resource Kit Documentation Set	32
<i>Tools Reference</i>	32
<i>LDAP SDK for C Programming Guide</i>	32
<i>LDAP SDK for Java Programming Guide</i>	33
Documentation Conventions Used in This Guide	33
Typographical Conventions	33
Terminology	33
Related Sources of Information	34
Directory Server Documentation	34
Sources of Information	34
Third-Party Sources of Information	35
The Mozilla Project	35
LDAP and Associated Information	35
Acknowledgements	36
Documentation Feedback	36
 Part 1 Getting Started	 37
Chapter 1 Introduction	39
Overview	39
The LDAP API	40

Synchronous and Asynchronous Operations	40
Installation Instructions	41
Provided Directories and Files	41
Header Files	42
ldap.h	43
lber.h	43
ldap_ssl.h	43
ldappr.h	44
Miscellaneous Files	44
Libraries	44
Library Naming Conventions	45
Installed Shared Libraries	45
Dependencies	45
LDAP Tools	46
Compiling Applications	46
Compiling on UNIX Platforms	47
Compiling on Windows Systems	47
Using the DLL	48
Definitions	48
Linking To NSPR Libraries	48
Linking Dependencies	48
The Example Programs	49
Synchronous Examples	49
Asynchronous Examples	50
 Chapter 2 Sample Client	 53
The Source Code	53
Compiling the Client	55
Compiling on Solaris	55
Compiling on Windows NT	55
Running the Client	56
Running on UNIX	56
Running on Windows NT	56
 Part 2 Working with the LDAP SDK for C	 57
 Chapter 3 Writing an LDAP Client	 59
Designing an LDAP Client	59
Initializing an LDAP Session	61
Specifying a Single LDAP Server	62
Specifying a List of LDAP Servers	62

Session Initialization Sample Code	63
Setting Session Preferences	64
Setting the Restart Preference	65
Specifying the LDAP Version of Your Client	65
Binding and Authenticating to an LDAP Server	66
Methods of Authentication	67
Simple Authentication	67
Certificate-based Client Authentication	67
Simple Authentication and Security Layer (SASL)	67
Using Simple Authentication	67
Performing a Synchronous Authentication Operation	68
Performing an Asynchronous Authentication Operation	69
Binding Anonymously	72
Performing LDAP Operations	73
Closing the Connection to the Server	73
 Chapter 4 Using the LDAP API	75
Synchronous and Asynchronous Functions	75
Calling Synchronous Functions	76
Calling Asynchronous Functions	77
To Verify That an LDAP Request Was Sent	77
To Retrieve the Server Response	78
To Get Information from a Server Response	80
To Free LDAPMessage Structure	82
To Cancel an Operation in Progress	82
Sample Code to Call an Asynchronous Function	82
Retrieving SDK Information	85
Managing Memory	86
Reporting Errors	87
Setting Error Codes	87
Getting Information About an Error	88
To Get Information from an LDAPMessage Structure	88
To Get Information from an LDAP Structure	90
Getting the Error Message from an Error Code	91
Receiving the Matching Portion of a DN	92
Printing Out Error Messages	92
Handling Referrals	93
Search References and Referrals	94
Enabling or Disabling Referral Handling	95
Limiting Referral Hops	95
Binding For Referrals	96
How Referral Binding Works	96
Defining the Rebind Function	97

Registering the Rebind Function	98
Setting Up an In-Memory Cache	98
Handling Failover	100
Creating a New Connection Handle	100
Using the Reconnect Option	101
 Chapter 5 Searching a LDAP Directory	105
Overview	105
Sending a Search Request	106
Search Parameters	107
Specifying the Base DN and Scope	108
Defining a Search Filter	109
Specifying Attributes to Retrieve	111
Sorting Attributes	112
Operational Attributes	112
Setting Search Preferences	112
Getting Search Results	114
Getting Results Synchronously	115
Getting Results Asynchronously	118
Determining Search Result Types	123
Getting Distinguished Names for Each Entry	125
Getting the Distinguished Name of an Entry	125
Getting the Components of a Distinguished Name	126
Getting Attribute Names from an Entry	127
Getting the Values of an Attribute	128
Getting Referrals from Search References	130
Sorting the Search Results	131
Server-Side Sorting	131
Client-Side Sorting	131
Sorting Entries by an Attribute	132
Sorting Entries by Multiple Attributes	132
Sorting the Values of an Attribute	133
Freeing the Search Results	134
Search Examples	134
Reading an Entry with a Search	134
Listing Subentries with a Search	136
Sending A Search Request	137
 Chapter 6 Using Filter Configuration Files	139
Filter Configuration Files	139
The Filter Configuration File Syntax	140
Tag	141

Patterns	141
Delimiters	141
Filters	141
Descriptions	142
Filter Parameters	142
Loading a Filter Configuration File	143
Retrieving Filters	143
Filter Prefixes and Suffixes	145
Freeing Filters from Memory	145
Creating Filters Programmatically	146
 Chapter 7 Adding, Modifying and Deleting Entries	147
Specifying Entry Information	147
Adding an Entry	149
Specifying Values for Attributes	149
Specifying a Single Value in an Attribute	149
Specifying Multiple Values in an Attribute	150
Specifying Binary Data as the Value of an Attribute	150
Specifying Attributes in the Entry	152
Adding the Entry to the Directory	153
Synchronous Add Operation	153
Asynchronous Add Operation	156
Modifying an Entry	161
Types of Modifications	162
Replacing the Values of an Attribute	162
Removing Values from an Attribute	163
Adding Values to an Attribute	164
Removing an Attribute	165
Adding an Attribute	165
Creating an Array of Changes	166
Modifying the Entry in the Directory	166
Synchronous Modify Operation	166
Asynchronous Modify Operation	169
Deleting an Entry	174
Synchronous Delete Operation	174
Asynchronous Delete Operation	176
Changing the DN of an Entry	181
Synchronous Renaming Operation	181
Asynchronous Renaming Operation	182
Deleting the Attribute from the Old RDN	184
Changing the Location of the Entry	185
Synchronous Relocation of an Entry	185
Asynchronous Relocation of an Entry	186

Chapter 8 Comparing Entries	189
Overview	189
Synchronous Comparison Operations	190
Asynchronous Comparison Operations	192

Chapter 9 Working with LDAP URLs	199
Understanding LDAP URLs	199
Escaping Characters	200
Delimiting Fields	200
Examples of LDAP URLs	201
Specifying a Base Search	201
Retrieving One Attribute	201
Retrieving Multiple Attributes	201
Retrieving Entries from Multiple Levels	202
Retrieving Entries from One Level	202
Determining an LDAP URL	202
Getting the Components of an LDAP URL	203
Freeing the Components of an LDAP URL	206
Processing an LDAP URL	207

Part 3 Advanced Topics **209**

Chapter 10 Retrieving Server Information	211
Understanding DSEs	211
Getting the Root DSE	212
Determining LDAPv3 Support	215
Getting Schema Information	217

Chapter 11 Connecting Over SSL	219
SSL and the LDAP SDK for C	219
Connecting to an LDAP Server Over SSL	221
Alternative to ldapssl_init()	222
Handling Errors	223
Installing Your Own SSL I/O Functions	223
Using Certificate-Based Client Authentication	223

Chapter 12 Using SASL Authentication	225
Understanding SASL	225
Determining Supported SASL Mechanisms	225
Authenticating Using SASL	226

Synchronous SASL Bind Operation	226
Asynchronous SASL Bind Operation	227
Chapter 13 Working with LDAP Controls	231
How LDAP Controls Work	231
Using Controls in the LDAP API	232
Determining Supported Controls	234
Using the Server-Side Sorting Control	237
Specifying the Sort Order	237
Creating the Control	238
Performing the Search	238
Interpreting the Results	239
Server-Sorting Control Sample Program	240
Using the Persistent Search Control	242
Using the Entry Change Notification Control	244
Using the Virtual List View Control	245
Using the Manage DSA IT Control	245
Using Password Policy Controls	245
Using the Proxied Authorization Control	246
The Proxy Right	247
The Proxied Authorization Control	247
Proxy Authorization Sample Program	248
Chapter 14 Writing Multithreaded Clients	251
Specifying Thread Functions	251
Setting Up the ldap_thread_fns Structure	251
Setting Up the ldap_extra_thread_fns Structure	253
Setting Session Options	254
Pthreads Client Application	255
Chapter 15 Extended Operations	267
How Extended Operations Work	267
Determining Supported Extended Operations	268
Performing an Extended Operation	268
Synchronous Extended Operation	268
Asynchronous Extended Operation	268
Sample Code to Perform an Extended Operation	269

Part 4 Reference Guide 271

Chapter 16 Data Type Reference 273

Conventions	273
Typographical	273
Deprecated Structures	274
Structure Summary by Task	274
BER Structures	274
Structures for the Core API	274
Structures for API Extensions	275
Referral Binding Callback	276
Client-Side Sorting Callbacks	276
Extended I/O Control	276
Memory Management Control	277
Thread Signaling Controls	278
Deprecated and Outdated Types	279
Structures Alphabetically	280

Chapter 17 Function Reference 327

Typographic Conventions	327
Function Summary by Task	328
Working with Basic Encoding Rules	328
Managing LDAP Sessions	330
Performing LDAP Operations	330
Processing Search Results	331
Reading the Contents of an Entry	332
Sorting Search Results	333
Working with Search Filters	334
Using LDAPv3 Controls	334
Using Sun Java System Extensions	335
Working with LDAP URLs	335
Working with UTF-8 Encoding	336
Handling Errors	337
Managing Memory	337
Working with Netscape Portable Runtime	338
Deprecated and Outdated Functions	339
Outdated Standard Functions and Equivalents	340
Functions Alphabetically	341

Chapter 18 Result Codes 635

Overview	635
Result Codes Summary in Numerical Order	635

Result Codes Summary By Server/Client	638
Result Codes Reference in Alphabetical Order	639
Glossary	659
Appendix A An Introduction to LDAP	663
Overview	663
How Directory Services Work	664
Defining Information Using LDAP	664
The LDAP Schema	665
The LDIF File Format	666
Organizing LDAP Within a Directory Server	667
Distinguished Names	667
Data Distribution	668
Additional Information	668
Index	669

List of Figures

Figure 5-1	Search Scope of LDAP_SCOPE_SUBTREE	108
Figure 5-2	Search Scope of LDAP_SCOPE_ONELEVEL	109
Figure 5-3	Search Scope of LDAP_SCOPE_BASE	109
Figure A-1	Sample Hierarchy of Entries	667
Figure A-2	DN to Identify Entries	668

List of Tables

Table 0-1	Additional Sources of Information	34
Table 1-1	LDAP SDK for C Directories	41
Table 1-2	LDAP SDK for C Header Files	42
Table 1-3	Sample Configuration Files	44
Table 1-4	Library Naming Convention by Operating System	45
Table 1-5	Shared Libraries	45
Table 1-6	LDAP Utilities	46
Table 1-7	Synchronous Example Programs	50
Table 1-8	Asynchronous Example Programs	50
Table 3-1	Performing LDAP Operations	73
Table 4-1	SDK Functions to Allocate and Free Memory	86
Table 4-2	LDAP_CALL and LDAP_CALLBACK parameters	97
Table 5-1	Search Criteria Parameters	107
Table 5-2	Search Filter Syntax	109
Table 5-3	Basic Operators for Search Filters	110
Table 5-4	Boolean Search Filter Syntax	110
Table 5-5	Boolean Operators for Search Filters	111
Table 5-6	Information Available in Operational Attributes	112
Table 6-1	Filter Parameters	142
Table 7-1	LDAPMod Field Descriptions	148
Table 7-2	berval Field Descriptions	151
Table 9-1	Components of an LDAP URL	199
Table 9-2	ldap_url_desc Field Descriptions	203
Table 10-1	Information available in the root DSE	212
Table 13-1	LDAPv3 Server Controls	234
Table 13-2	Server-side Sorting Result Codes	239
Table 13-3	Server Responses to Sorting Controls	239

Table 14-1	ldap_thread_fns Field Descriptions	252
Table 16-1	Typographical Conventions in Data Types and Structures	273
Table 16-2	BER Structures	274
Table 16-3	Core API Structures	275
Table 16-4	API Extension Structures	275
Table 16-5	Referral Binding Data Type	276
Table 16-6	Client-side Sorting Structures	276
Table 16-7	Structures and Prototypes for Extended I/O Functionality	277
Table 16-8	Memory Management Datatypes	277
Table 16-9	Thread Signaling Control Data Types	278
Table 16-10	Deprecated Data Types and Their Replacements	279
Table 16-11	Outdated Types and Their New Equivalent	279
Table 16-12	berval Field Descriptions	281
Table 16-13	LDAPAPIFeatureInfo Structure Parameters	283
Table 16-14	LDAPAPIInfo Structure Parameters	284
Table 16-15	LDAPControl Field Descriptions	288
Table 16-16	ldap_dns_fns Field Descriptions	290
Table 16-17	ldap_extra_thread_fns Field Descriptions	292
Table 16-18	LDAPFilterInfo Field Descriptions	293
Table 16-19	LDAPHostEnt Field Descriptions	296
Table 16-20	ldap_io_fns Field Descriptions	300
Table 16-21	Functions for LDAPMemCache Structure	303
Table 16-22	LDAPMod Field Descriptions	305
Table 16-23	LDAPsortkey Field Descriptions	308
Table 16-24	ldap_thread_fns Field Descriptions	314
Table 16-25	LDAPURLDesc Field Descriptions	316
Table 16-26	LDAPURLDesc Parsed URL Values	317
Table 16-27	LDAPVersion Field Definitions	318
Table 16-28	LDAPVirtualList Field Descriptions	321
Table 17-1	Function Naming Conventions	327
Table 17-2	Functions to Encode and Decode BER	328
Table 17-3	Functions to Manage an LDAP Session	330
Table 17-4	Functions to Perform Operations on LDAP Server	330
Table 17-5	Functions to Process Search Results	331
Table 17-6	Functions to Access Contents of an Entry	332
Table 17-7	Client-side Sorting Functions	333
Table 17-8	Server-side Sorting Functions	333

Table 17-9	Functions to Initialize, Retrieve, and Build Filters	334
Table 17-10	Functions to Manage LDAPv3 Controls	335
Table 17-11	Sun Java System Extensions	335
Table 17-12	Functions to Interpret LDAP URLs	336
Table 17-13	Functions to Handle UTF-8 Encoding	336
Table 17-14	Functions for Error Handling	337
Table 17-15	Functions to Free Memory	337
Table 17-16	Functions to Directly Manage Memory	338
Table 17-17	Functions to Work with Netscape Portable Runtime	338
Table 17-18	Deprecated Functions and Their Replacement	339
Table 17-19	Outdated LDAPv2 Functions and New Equivalent	340
Table 17-20	ber_alloc_t() Function Parameter	342
Table 17-21	ber_bvecfree() Function Parameter	343
Table 17-22	ber_bvdup() Function Parameter	343
Table 17-23	ber_bvfree() Function Parameter	344
Table 17-24	ber_dup() Function Parameter	344
Table 17-25	ber_first_element() Function Parameters	345
Table 17-26	ber_flatten() Function Parameters	346
Table 17-27	ber_free() Function Parameters	347
Table 17-28	ber_get_boolean() Function Parameters	348
Table 17-29	ber_get_int() Function Parameters	349
Table 17-30	ber_get_next() Function Parameters	349
Table 17-31	ber_get_next_buffer_ext() Function Parameters	350
Table 17-32	ber_get_null() Function Parameter	351
Table 17-33	ber_init() Function Parameter	352
Table 17-34	ber_next_element() Function Parameters	353
Table 17-35	ber_peek_tag() Function Parameters	354
Table 17-36	ber_printf() Function Parameters	355
Table 17-37	ber_put_seq() Function Parameter	357
Table 17-38	ber_put_set() Function Parameter	358
Table 17-39	ber_scanf() Function Parameters	359
Table 17-40	ber_skip_tag() Function Parameters	361
Table 17-41	ldap_abandon() Function Parameters	365
Table 17-42	ldap_abandon_ext() Function Parameters	367
Table 17-43	ldap_add() Function Parameters	369
Table 17-44	ldap_add_ext() Function Parameters	371
Table 17-45	ldap_add_ext_s() Function Parameters	373

Table 17-46	ldap_add_s() Function Parameters	376
Table 17-47	ldap_ber_free() Function Parameters	378
Table 17-48	ldap_bind() Function Parameters	378
Table 17-49	ldap_bind_s() Function Parameters	379
Table 17-50	ldap_compare() Function Parameters	382
Table 17-51	ldap_compare_ext() Function Parameters	384
Table 17-52	ldap_compare_ext_s() Function Parameters	386
Table 17-53	ldap_compare_s() Function Parameters	388
Table 17-54	ldap_control_free() Function Parameter	389
Table 17-55	ldap_controls_free() Function Parameter	390
Table 17-56	ldap_count_entries() Function Parameters	390
Table 17-57	ldap_count_messages() Function Parameters	391
Table 17-58	ldap_count_references() Function Parameters	393
Table 17-59	ldap_count_values() Function Parameter	394
Table 17-60	ldap_count_values_len() Function Parameter	395
Table 17-61	ldap_create_filter() Function Parameters	397
Table 17-62	ldap_create_persistentsearch_control() Function Parameters	398
Table 17-63	ldap_create_geteffectiveRights_control() Function Parameters	401
Table 17-64	ldap_create_proxiedauth_control() Function Parameters	402
Table 17-65	ldap_create_proxyauth_control() Function Parameters	403
Table 17-66	ldap_create_sort_control() Function Parameters	404
Table 17-67	ldap_create_sort_keylist() Function Parameters	406
Table 17-68	ldap_create_virtuallist_control() Function Parameters	407
Table 17-69	ldap_delete() FunctionParameters	409
Table 17-70	ldap_delete_ext() Function Parameters	410
Table 17-71	ldap_delete_ext_s() Function Parameters	412
Table 17-72	ldap_delete_s() Function Parameters	414
Table 17-73	ldap_dn2ufn() Function Parameter	415
Table 17-74	ldap_entry2html() Function Parameters	416
Table 17-75	ldap_entry2html_search() Function Parameters	418
Table 17-76	ldap_entry2text() Function Parameters	420
Table 17-77	ldap_entry2text_search() Function Parameters	421
Table 17-78	ldap_err2string() Function Parameter	423
Table 17-79	ldap_explode_dn() Function Parameters	424
Table 17-80	ldap_explode_dns() Function Parameter	426
Table 17-81	ldap_explode_rdn() Function Parameters	426
Table 17-82	ldap_extended_operation() Function Parameters	427

Table 17-83	ldap_extended_operation_s() Function Parameters	429
Table 17-84	ldap_first_attribute() Function Parameters	431
Table 17-85	ldap_first_disptmpl() Function Parameter	433
Table 17-86	ldap_first_entry() Function Parameters	434
Table 17-87	ldap_first_message() Function Parameters	435
Table 17-88	ldap_first_reference() Function Parameters	437
Table 17-89	ldap_first_searchobj() Function Parameter	438
Table 17-90	ldap_first_tmplcol() Function Parameters	439
Table 17-91	ldap_first_tmplrow() Function Parameter	439
Table 17-92	ldap_free_friendlymap() Function Parameter	440
Table 17-93	ldap_free_searchprefs() Function Parameter	441
Table 17-94	ldap_free_sort_keylist() Function Parameter	442
Table 17-95	ldap_free_templates() Function Parameter	442
Table 17-96	ldap_free_urldesc() Function Parameter	443
Table 17-97	ldap_friendly_name() function parameters	444
Table 17-98	ldap_get_dn() Function Parameters	446
Table 17-99	ldap_get_entry_controls() Function Parameters	448
Table 17-100	ldap_getfilter_free() Function Parameter	449
Table 17-101	ldap_getfirstfilter() Function Parameters	450
Table 17-102	ldap_get_lang_values() Function Parameters	453
Table 17-103	ldap_get_lang_values_len() Function Parameters	454
Table 17-104	ldap_get_lderrno() Function Parameters	455
Table 17-105	ldap_getnextfilter() Function Parameter	457
Table 17-106	ldap_get_option() Function Parameters	458
Table 17-107	Options for the ldap_get_options() Function	458
Table 17-108	ldap_get_values() Function Parameters	463
Table 17-109	ldap_get_values_len() Function Parameters	465
Table 17-110	ldap_init() Function Parameters	467
Table 17-111	ldap_init_getfilter() Function Parameter	470
Table 17-112	ldap_init_getfilter_buf() Function Parameters	471
Table 17-113	ldap_init_searchprefs() Function Parameters	473
Table 17-114	ldap_init_searchprefs_buf() Function Parameters	474
Table 17-115	ldap_init_templates() Function Parameters	475
Table 17-116	ldap_init_templates_buf() Function Parameters	476
Table 17-117	ldap_is_ldap_url() Function Parameter	477
Table 17-118	ldap_keysort_entries() Function Parameter	478
Table 17-119	ldap_memcache_destroy() Function Parameter	479

Table 17-120	ldap_memcache_flush() Function Parameters	480
Table 17-121	ldap_memcache_get() Function Parameters	481
Table 17-122	ldap_memcache_init() Function Parameters	482
Table 17-123	ldap_memcache_set() Function Parameters	483
Table 17-124	ldap_memcache_update() Function Parameter	484
Table 17-125	ldap_memfree() Function Parameter	485
Table 17-126	ldap_modify() Function Parameters	486
Table 17-127	ldap_modify_ext() Function Parameters	489
Table 17-128	ldap_modify_ext_s() Function Parameters	491
Table 17-129	ldap_modify_s() Function Parameters	493
Table 17-130	ldap_modrdn2() Function Parameters	495
Table 17-131	ldap_modrdn2_s() Function Parameters	497
Table 17-132	ldap_mods_free() Function Parameters	499
Table 17-133	ldap_msgfree() Function Parameter	500
Table 17-134	ldap_msgid() Function Parameter	502
Table 17-135	ldap_msgtype() Function Parameter	503
Table 17-136	ldap_multisort_entries() Function Parameters	505
Table 17-137	ldap_name2template() Function Parameters	507
Table 17-138	ldap_next_attribute() Function Parameters	508
Table 17-139	ldap_next_disptmpl() Function Parameters	509
Table 17-140	ldap_next_entry() Function Parameters	510
Table 17-141	ldap_next_message() Function Parameters	511
Table 17-142	ldap_next_reference() Function Parameters	512
Table 17-143	ldap_next_searchobj() Function Parameters	513
Table 17-144	ldap_next_tmplcol() Function Parameters	514
Table 17-145	ldap_next_tmplrow() Function Parameters	515
Table 17-146	ldap_oc2template() Function Parameters	516
Table 17-147	ldap_open() Function Parameters	517
Table 17-148	ldap_parse_entrychange_control() Function Parameters	518
Table 17-149	ldap_parse_extended_result() Function Parameters	519
Table 17-150	ldap_parse_reference() Function Parameters	521
Table 17-151	ldap_parse_result() Function Parameters	523
Table 17-152	ldap_parse_sasl_bind_result() Function Parameters	525
Table 17-153	ldap_parse_sort_control() Function Parameters	527
Table 17-154	ldap_parse_virtuallist_control() Function Parameters	529
Table 17-155	ldap_perror() Function Parameters	531
Table 17-156	ldap_rename() Function Parameters	532

Table 17-157	ldap_rename() Function Parameters	534
Table 17-158	ldap_result() Function Parameters	537
Table 17-159	ldap_result2error() Function Parameters	539
Table 17-160	ldap_sasl_bind() Function Parameters	540
Table 17-161	ldap_sasl_bind_s() Function Parameters	542
Table 17-162	ldap_search() Function Parameters	544
Table 17-163	ldap_search_ext() Function Parameters	548
Table 17-164	ldap_search_ext_s() Function Parameters	551
Table 17-165	ldap_search_s() Function Parameters	553
Table 17-166	ldap_search_st() Function Parameters	556
Table 17-167	ldap_set_filter_additions() Function Parameters	557
Table 17-168	ldap_set_lderrno() Function Parameters	559
Table 17-169	ldap_set_option() Function Parameters	560
Table 17-170	Options for ldap_set_option()	561
Table 17-171	ldap_set_rebind_proc() Function Parameters	566
Table 17-172	ldap_simple_bind() Function Parameters	570
Table 17-173	ldap_simple_bind_s() Function Parameters	572
Table 17-174	ldap_sort_entires() Function Parameters	574
Table 17-175	ldap_sort_strcasecmp() Function Parameters	576
Table 17-176	ldap_sort_values() Function Parameters	577
Table 17-177	ldapssl_advcliauth_init() Function Parameters	579
Table 17-178	ldapssl_client_init() Function Parameters	581
Table 17-179	ldapssl_cliauth_init() Function Parameters	583
Table 17-180	ldapssl_enable_cliauth() Function Parameters	585
Table 17-181	ldapssl_err2string() Function Parameters	585
Table 17-182	ldapssl_init() Function Parameters	587
Table 17-183	ldapssl_install_routines() Function Parameter	588
Table 17-184	ldapssl_serverauth_init() Function Parameters	592
Table 17-185	ldapssl_set_strength() Function Parameters	593
Table 17-186	ldapssl_tls_start_s() Function Parameter	594
Table 17-187	ldap_tmplattrs() Function Parameters	595
Table 17-188	ldap_tmplerr2string() Function Parameter	596
Table 17-189	ldap_unbind() Function Parameter	597
Table 17-190	ldap_unbind_s() Function Parameter	598
Table 17-191	ldap_unbind_ext() Function Parameters	600
Table 17-192	ldap_url_parse() Function Parameters	601
Table 17-193	ldap_url_parse_no_defaults() Function Parameters	604

Table 17-194	ldap_url_search() Function Parameters	605
Table 17-195	ldap_url_search_s() Function Parameters	608
Table 17-196	ldap_url_search_st()Function Parameters	610
Table 17-197	ldap_utf8characters()Function Parameter	611
Table 17-198	ldap_utf8copy()Function Parameter	612
Table 17-199	ldap_utf8getcc()Function Parameter	613
Table 17-200	ldap_utf8isalnum()Function Parameter	613
Table 17-201	ldap_utf8isalpha()Function Parameter	613
Table 17-202	ldap_utf8isdigit()Function Parameter	614
Table 17-203	ldap_utf8isspace()Function Parameter	614
Table 17-204	ldap_utf8len()Function Parameter	615
Table 17-205	ldap_utf8next()Function Parameter	615
Table 17-206	ldap_utf8prev()Function Parameter	616
Table 17-207	ldap_utf8strtok_r() Function Parameter	616
Table 17-208	ldap_vals2html() Function Parameters	617
Table 17-209	ldap_vals2text() Function Parameters	618
Table 17-210	ldap_value_free() Function Parameter	619
Table 17-211	ldap_value_free_len() Function Parameter	620
Table 17-212	ldap_version() Function Parameter	621
Table 17-213	ldap_x_calloc() Function Parameters	622
Table 17-214	ldap_x_free() Function Parameter	622
Table 17-215	ldap_x_malloc() Function Parameter	624
Table 17-216	ldap_x_realloc() Function Parameter	625
Table 17-217	prldap_get_default_socket_info() Function Parameters	625
Table 17-218	prldap_get_session_info() Function Parameters	626
Table 17-219	prldap_get_socket_info() Function Parameters	628
Table 17-220	prldap_init() Function Parameters	628
Table 17-221	prldap_install_routines() Function Parameters	629
Table 17-222	prldap_set_default_socket_info() Function Parameters	630
Table 17-223	prldap_set_session_info() Function Parameters	631
Table 17-224	prldap_set_session_option() Function Parameters	631
Table 17-225	prldap_set_socket_info() Function Parameters	632
Table 18-1	Numerical Listing of Result Codes by Value	636
Table 18-2	LDAP Server Result Codes	638
Table 18-3	LDAP Client Result Codes	639

List of Code Examples

Code Example 2-1	Source Code to Retrieve LDAP Entry	53
Code Example 2-2	Solaris Makefile for Sample Client	55
Code Example 3-1	Source Code for LDAP search operation	60
Code Example 3-2	Passing an LDAP Server with the Default LDAP Port	62
Code Example 3-3	Passing Multiple LDAP Servers with Default LDAP Port	63
Code Example 3-4	Passing Non-Default LDAP Ports	63
Code Example 3-5	Initializing an LDAP Session	64
Code Example 3-6	Passing Restart Preferences to Function	65
Code Example 3-7	Passing the LDAP Protocol Version Number	66
Code Example 3-8	Synchronous Authentication	68
Code Example 3-9	Asynchronous Authentication	70
Code Example 3-10	Binding Anonymously	72
Code Example 3-11	Closing an LDAP Server Connection	74
Code Example 4-1	Calling Synchronous ldap_delete_ext_s to Delete an Entry	76
Code Example 4-2	Calling Asynchronous ldap_delete_ext and Send Verification	77
Code Example 4-3	Sample Loop to Poll for Results	79
Code Example 4-4	While Loop to Call Function	79
Code Example 4-5	Calling ldap_parse_result() Function	80
Code Example 4-6	Retrieve Error Information from LDAPMessage Structure	81
Code Example 4-7	Asynchronous Function to Delete Entry Code	82
Code Example 4-8	Retrieving SDK Version Information with ldap_get_option()	85
Code Example 4-9	Setting an Error Code	87
Code Example 4-10	Calling ldap_parse_result() Function	88
Code Example 4-11	Get and Print Error Information from LDAPMessage Structure	89
Code Example 4-12	Calling ldap_get_lderrno() Function	90
Code Example 4-13	To Get Error Message from an LDAP Structure	91
Code Example 4-14	Retrieving an Error Message from an Error Code	92

Code Example 4-15	Printing Error Codes	92
Code Example 4-16	Disabling Referrals	95
Code Example 4-17	Rebind Function Definition	97
Code Example 4-18	Creating an In-Memory Cache	99
Code Example 4-19	Initializing a New Connection Handle	100
Code Example 4-20	Reconnecting to an Existing Connection Handle	101
Code Example 4-21	Reconnecting to Server After Disconnect	102
Code Example 5-1	ldap_search_ext_s() Search Request	107
Code Example 5-2	ldap_search_ext() Search Request	107
Code Example 5-3	Sample Code to Set Session Search Preferences	113
Code Example 5-4	Synchronous Searching	115
Code Example 5-5	Declaring ldap_result()	118
Code Example 5-6	Retrieving Search Results One at a Time	118
Code Example 5-7	Printing Results of Asynchronous Search	119
Code Example 5-8	Retrieving a Chain of Results	124
Code Example 5-9	Obtaining DN for Search Result Entries	126
Code Example 5-10	Retrieving Entry Attributes	127
Code Example 5-11	Retrieving Attribute Values	128
Code Example 5-12	Getting and Saving Attribute Value	129
Code Example 5-13	Obtaining a Referral	130
Code Example 5-14	Sorting Entries by Attribute Value	132
Code Example 5-15	Sorting Entries by Multiple Values	132
Code Example 5-16	Sorting Attribute Values	133
Code Example 5-17	Reading a Specific Entry with a Search	134
Code Example 5-18	Listing Subentries	136
Code Example 5-19	Sample Code to Send a Search Request	137
Code Example 6-1	Section of Filter Configuration File	140
Code Example 6-2	Syntax for Filter Configuration File	140
Code Example 6-3	Retrieving Configuration Filters	143
Code Example 6-4	Adding Prefixes and Suffixes	145
Code Example 6-5	Freeing Filters from Memory	146
Code Example 6-6	Creating Filters	146
Code Example 7-1	Defining a LDAPMod Structure	148
Code Example 7-2	Setting Up an Attribute Structure	150
Code Example 7-3	Specifying Multiple Values in an Attribute	150
Code Example 7-4	Berval Structure	151
Code Example 7-5	Adding a Value to an Attribute	151

Code Example 7-6	Adding Array of Structures to an Attribute	152
Code Example 7-7	Synchronous Add Operation	153
Code Example 7-8	Sample Synchronous Add Entry Program	154
Code Example 7-9	Asynchronous Add Operation	157
Code Example 7-10	Sample Asynchronous Add Program	158
Code Example 7-11	Modifying an Attribute Value	163
Code Example 7-12	Removing an Attribute Value	163
Code Example 7-13	Adding Values to an Attribute	164
Code Example 7-14	Assembling an Array of LDAPMod Structures	166
Code Example 7-15	Synchronous Modify Operation	167
Code Example 7-16	Sample Synchronous Modify Program	167
Code Example 7-17	Asynchronous Modify Operation	170
Code Example 7-18	Sample Asynchronous Modify Program	171
Code Example 7-19	Synchronous Delete Operation	175
Code Example 7-20	Sample Synchronous Delete Program	175
Code Example 7-21	Asynchronous Delete Operation	177
Code Example 7-22	Sample Asynchronous Delete Program	178
Code Example 7-23	Synchronous Renaming Operation	181
Code Example 7-24	Asynchronous Renaming Operation	183
Code Example 7-25	Synchronous Relocating of an Entry	185
Code Example 7-26	Asynchronous Relocating of an Entry	186
Code Example 8-1	Synchronous Compare Operation	190
Code Example 8-2	Sample Synchronous Comparison Program	191
Code Example 8-3	Asynchronous Compare Operation	193
Code Example 8-4	Sample Asynchronous Comparison Program	195
Code Example 9-1	Determining an LDAP URL	202
Code Example 9-2	LDAPURLDesc Structure	203
Code Example 9-3	Parsing an LDAP URL	204
Code Example 9-4	Results of Parsing an LDAP URL	205
Code Example 9-5	Freeing LDAPURLDesc Structure From Memory	206
Code Example 9-6	Processing an LDAP URL Search Request	207
Code Example 10-1	Getting the Root DSE Attribute Values	213
Code Example 10-2	Determining the Supported LDAP Version	215
Code Example 11-1	Initializing a Client SSL Connection	221
Code Example 11-2	An Alternate Client SSL Initialization	222
Code Example 11-3	Setting Custom SSL I/O Functions	223
Code Example 12-1	Authenticating over SASL	228

Code Example 13-1	Example of LDAPControl Structure	233
Code Example 13-2	Sample Program to Search for Supported Controls	235
Code Example 13-3	Sample Program to Apply Server-sorting Control	240
Code Example 13-4	Calling ldap_create_persistentsearch_control()	243
Code Example 13-5	Proxy Right ACI	247
Code Example 13-6	Proxied Authorization Control Example	248
Code Example 14-1	ldap_thread_fns Structure	252
Code Example 14-2	Declaring ldap_extra_thread_fns Structure	253
Code Example 14-3	Setting Up an ldap_thread_fns Structure	254
Code Example 14-4	Using POSIX Threading Under Solaris	255
Code Example 15-1	Requesting an Extended Operation	269
Code Example 16-1	berval Definition	280
Code Example 16-2	LDAPAPIFeatureInfo Definition	282
Code Example 16-3	Sequence to Retrieve LDAPAPIFeatureInfo	283
Code Example 16-4	LDAPAPIInfo Definition	283
Code Example 16-5	Sequence to Retrieve LDAPAPIInfo	284
Code Example 16-6	LDAP_CMP_CALLBACK Prototype	287
Code Example 16-7	Definition of LDAPControl	288
Code Example 16-8	LDAP_DNSFN_GETHOSTBYADDR Prototype	289
Code Example 16-9	LDAP_DNSFN_GETHOSTBYNAME Prototype	289
Code Example 16-10	ldap_dns_fns Definition	290
Code Example 16-11	ldap_extra_thread_fns Definition	291
Code Example 16-12	LDAPFiltInfo Definition	293
Code Example 16-13	Sample LDAPFiltInfo Code	294
Code Example 16-14	LDAPHostEnt Definition	295
Code Example 16-15	LDAP_IOF_CLOSE_CONNECT_CALLBACK Prototype	296
Code Example 16-16	LDAP_IOF_CONNECT_CALLBACK Prototype	297
Code Example 16-17	LDAP_IOF_IOCTL_CALLBACK Prototype	297
Code Example 16-18	LDAP_IOF_READ_CALLBACK Prototype	297
Code Example 16-19	LDAP_IOF_SELECT_CALLBACK Prototype	298
Code Example 16-20	LDAP_IOF_SOCKET_CALLBACK Prototype	298
Code Example 16-21	LDAP_IOF_SSL_ENABLE_CALLBACK Prototype	299
Code Example 16-22	LDAP_IOF_WRITE_CALLBACK Prototype	299
Code Example 16-23	ldap_io_fns Structure	299
Code Example 16-24	Installing ldap_memalloc_fns	302
Code Example 16-25	LDAPMod Definition	304
Code Example 16-26	Setting Up an LDAPMod Structure	306

Code Example 16-27	LDAP_REBINDPROC_CALLBACK Prototype	307
Code Example 16-28	LDAPsortkey Definition	308
Code Example 16-29	LDAP_TF_GET_ERRNO_CALLBACK Prototype	309
Code Example 16-30	LDAP_TF_GET_LDERRNO_CALLBACK Prototype	309
Code Example 16-31	LDAP_TF_SET_ERRNO_CALLBACK Prototype	310
Code Example 16-32	LDAP_TF_SET_LDERRNO_CALLBACK Prototype	310
Code Example 16-33	LDAP_TF_MUTEX_ALLOC_CALLBACK Prototype	311
Code Example 16-34	LDAP_TF_MUTEX_FREE_CALLBACK Prototype	311
Code Example 16-35	LDAP_TF_MUTEX_LOCK_CALLBACK Prototype	312
Code Example 16-36	LDAP_TF_MUTEX_UNLOCK_CALLBACK Prototype	312
Code Example 16-37	LDAP_TF_THREADID_CALLBACK Prototype	313
Code Example 16-38	ldap_thread_fns Structure Definition	314
Code Example 16-39	LDAPURLDesc Definition	316
Code Example 16-40	LDAP_VALCMP_CALLBACK Prototype	317
Code Example 16-41	LDAPVersion Definition	318
Code Example 16-42	LDAPVirtualList Definition	320
Code Example 16-43	LDAP_X_EXTIOF_CLOSE_CALLBACK Prototype	321
Code Example 16-44	LDAP_X_EXTIOF_CONNECT_CALLBACK Prototype	322
Code Example 16-45	LDAP_X_EXTIOF_DISPOSEHANDLE_CALLBACK Prototype	322
Code Example 16-46	LDAP_X_EXTIOF_NEWHANDLE_CALLBACK Prototype	323
Code Example 16-47	ldap_x_ext_io_fns Definition	323
Code Example 16-48	LDAP_X_EXTIOF_POLL_CALLBACK Prototype	324
Code Example 16-49	ldap_x_hostlist_status Definition	324
Code Example 16-50	LDAP_X_PollFD Structure Definition	325
Code Example 17-1	ber_free() Code Example	347
Code Example 17-2	Cancelling an ldap_url_search() Operation	366
Code Example 17-3	ldap_abandon_ext() Code Example	368
Code Example 17-4	ldap_add() Code Example	369
Code Example 17-5	ldap_add_s Code Example	376
Code Example 17-6	ldap_compare() Code Sample	383
Code Example 17-7	ldap_compare_s() Code Sample	388
Code Example 17-8	ldap_count_values() Code Example	394
Code Example 17-9	ldap_count_values_len() Code Example	396
Code Example 17-10	Creating a Filter with ldap_create_filter()	397
Code Example 17-11	ldap_delete() Code Example	409
Code Example 17-12	ldap_delete_s() Code Example	415
Code Example 17-13	ldap_err2string() Code Example	423

Code Example 17-14	ldap_first_attribute() Code Example	432
Code Example 17-15	ldap_free_friendlymap() Code Example	440
Code Example 17-16	ldap_free_urldesc() Code Example	443
Code Example 17-17	Unfriendly to Friendly Name Mapping File	445
Code Example 17-18	ldap_friendly_name() Code Example	445
Code Example 17-19	ldap_get_dn() Code Example	447
Code Example 17-20	ldap_getfilter_free() Code Example	449
Code Example 17-21	ldap_getfirstfilter() Code Example	451
Code Example 17-22	ldap_get_lderrno() Code Example	456
Code Example 17-23	ldap_get_option() Code Example	462
Code Example 17-24	LDAP_OPT_API_FEATURE_INFO and LDAP_OPT_API_INFO Usage ..	463
Code Example 17-25	ldap_get_values() Code Example	464
Code Example 17-26	ldap_get_values_len() Code Example	466
Code Example 17-27	Space-Delimited List for ldap_init()	468
Code Example 17-28	ldap_init() Example Using Variable Argument	469
Code Example 17-29	ldap_init() Code Example	469
Code Example 17-30	Loading a Filter Configuration File	471
Code Example 17-31	ldap_init_getfilter() Code Example	472
Code Example 17-32	ldap_is_ldap_url() Code Example	478
Code Example 17-33	ldap_memfree() Code Example	486
Code Example 17-34	ldap_modify() Code Example	487
Code Example 17-35	ldap_modify_s() Code Example	494
Code Example 17-36	ldap_modrdn2() Code Example	496
Code Example 17-37	ldap_modrdn2_s() Code Example	498
Code Example 17-38	ldap_mods_free() Code Example	499
Code Example 17-39	ldap_msgfree() Code Example	501
Code Example 17-40	ldap_msgid() Code Example	502
Code Example 17-41	ldap_msgtype() Code Example	504
Code Example 17-42	ldap_multisort_entries() Code Example	506
Code Example 17-43	ldap_perror() Code Example	531
Code Example 17-44	ldap_search() Code Example	545
Code Example 17-45	ldap_search_s() Code Example	554
Code Example 17-46	ldap_set_filter_additions() Code Example	558
Code Example 17-47	ldap_set_lderrno() Code Example	560
Code Example 17-48	ldap_set_option() Code Example	565
Code Example 17-49	ldap_set_rebind_proc() Code Example	568
Code Example 17-50	ldap_simple_bind() Code Example	571

Code Example 17-51	ldap_simple_bind_s() Code Example	574
Code Example 17-52	ldap_sort_entries() Code Example	575
Code Example 17-53	ldap_sort_values() Code Example	578
Code Example 17-54	ldapssl_advclientauth_init() Code Example	581
Code Example 17-55	ldapssl_client_init() Code Example	582
Code Example 17-56	ldapssl_clientauth_init() Code Example	584
Code Example 17-57	ldapssl_init() Code Example	587
Code Example 17-58	ldapssl_install_routines() Code Example	589
Code Example 17-59	ldapssl_pkcs_fns Structure Definition	590
Code Example 17-60	ldap_unbind() Code Example	598
Code Example 17-61	ldap_unbind_s() Code Example	599
Code Example 17-62	ldap_url_parse() Code Example	602
Code Example 17-63	ldap_url_search() Code Example	605
Code Example 17-64	ldap_url_search_s() Code Example	609
Code Example A-1	Sample LDAP Entry	664
Code Example A-2	Entry for Jennifer Jensen	665
Code Example A-3	Generic LDIF Format	666

Preface

The Sun™ Java System Directory Server Resource Kit *LDAP SDK for C Programming Guide* documents the Sun Java System LDAP SDK for C, a software development kit (SDK) for writing Lightweight Directory Access Protocol (LDAP) client applications. This preface contains the following sections:

- Audience for This Guide
- Directory Server Resource Kit Documentation Set
- Documentation Conventions Used in This Guide
- Related Sources of Information
- Acknowledgements
- Documentation Feedback

Audience for This Guide

This *LDAP SDK for C Programming Guide* is intended for use by C and C++ programmers who want to enable new or existing client applications to connect to, search or update databases that are managed by LDAP servers located on a network or the Internet. This book assumes that you are a programmer familiar with writing and compiling C applications on the platform(s) for which you want to implement this functionality. It is also recommended that programmers attempting this implementation understand the LDAP Data Interchange Format (LDIF).

In addition, programmers should be familiar with Sun Java System Directory Server, Sun Microsystems' LDAP data repository, and the command-line syntax for the particular shell and platform on which the server is installed. The books and guides described in "Related Sources of Information" on page 34 provide a good introduction to this information.

Directory Server Resource Kit Documentation Set

The Sun Java System DSRK documentation set contains three manuals:

- *Tools Reference*
- *LDAP SDK for C Programming Guide*
- *LDAP SDK for Java Programming Guide*

Tools Reference

The *Tools Reference* explains how to use each of the tools shipped with the DSRK. The book details their command-line syntax, listing all options, functionality, and return values (where applicable), as well as providing examples of usage.

LDAP SDK for C Programming Guide

The *LDAP SDK for C Programming Guide* (this guide) documents the LDAP SDK for C by explaining how to use it to enable applications to interact with LDAP servers. It offers a references to the LDAP API, which consists of data structures and functions. This manual is divided into four parts:

- Part 1, "Getting Started" explains how you can use the LDAP SDK for C to enable your applications for LDAP.
- Part 2, "Working with the LDAP SDK for C" explains how you can use the LDAP SDK for C to enable your applications for LDAP communications.
- Part 3, "Advanced Topics" contains additional material including descriptions of entries and attributes.
- Part 4, "Reference Guide" describes each data structure, function, and status code in the LDAP API.

LDAP SDK for Java Programming Guide

The *LDAP SDK for Java Programming Guide* documents the LDAP SDK for Java. It includes descriptions of, and reference material for, the interfaces, classes, and methods used to enable Java applications with the functionality to interact and communicate with LDAP servers. Also available are the Java API specifications or Javadocs.

Documentation Conventions Used in This Guide

In the DSRK documentation, certain typographical conventions and terminology are used. These are described in the following sections.

Typographical Conventions

This book uses the following typographical conventions:

- *Italic type* is used within text for book titles, new terminology, emphasis, and words used in the literal sense.
- Monospace font is used for sample code and code listings, API and language elements (such as function names and class names), filenames, pathnames, directory names, HTML tags, and any text that must be typed on the screen.
- *Italic serif font* is used within code and code fragments to indicate variable placeholders. For example, the following command uses *filename* as a variable placeholder for an argument to the `gunzip` command:

```
gunzip -d filename.tar.gz
```

NOTE Notes, Cautions and Tips highlight important conditions or limitations. Be sure to read this information before continuing.

Terminology

Below is a list of general terms used in the Directory Server Resource Kit documentation set:

- *DSRK* refers to the installed Directory Server Resource Kit software.

- *Directory Server* refers to an installed instance of Sun Java System Directory Server.
- *DSRK_base* is a variable place holder for the home directory where the Directory Server Resource Kit is installed.
- *DirectoryServer_base* is a variable place holder for the home directory where Sun Java System Directory Server is installed.

Related Sources of Information

In addition to the documentation provided with Directory Server Resource Kit, there are several other sources of information that might be helpful.

Directory Server Documentation

The tools in the DSRK interact with Sun Java System Directory Server. The following documentation is referenced throughout this guide:

- *Sun ONE Directory Server Deployment Guide*
- *Sun ONE Directory Server Installation and Tuning Guide*
- *Sun ONE Directory Server Administration Guide*
- *Sun ONE Directory Server Reference Manual*

A listing of the full set of documentation can be found at:

http://docs.sun.com/coll/S1_DirectoryServer_52

Sources of Information

Sun Microsystems has other helpful sources of information. Table 0-1 lists these sources.

Table 0-1 Additional Sources of Information

Source	Location
Sun Java System Download Center	http://www.sun.com/software/download/

Table 0-1 Additional Sources of Information (*Continued*)

Source	Location
Sun Java System Technical Support	http://www.sun.com/service/sunone/software/index.html
Sun Java System Professional Services Information	http://www.sun.com/service/sunps/sunone/index.html
Sun Enterprise Services, Solaris Patches and Support	http://sunsolve.sun.com/
JavaServer Pages	http://java.sun.com/products/jsp/
The Source for Developers	http://developers.sun.com/
Sun Training and Certification	http://suned.sun.com/

Third-Party Sources of Information

The following sections list sources of information found on sites that are not under the auspices of Sun Microsystems.

NOTE	Sun is not responsible for the availability of third-party Web sites mentioned in this document. Sun does not endorse nor is responsible or liable for any content, advertising, products, or other material available from such sites or resources. Sun will not be responsible or liable for any damage or loss caused by, alleged to be caused by, or in connection with, the use of or reliance on any such content, goods, or services that are available on or through any such sites or resources.

The Mozilla Project

SDKs for accessing LDAP in C and Java are also released through the Mozilla™ open source project. Instructions on obtaining the source code and contributing to the development of these API is available at:

<http://www.mozilla.org/directory>

LDAP and Associated Information

There are many sources of information detailing LDAP and its associated technologies.

NOTE	Note that some of the links provided are time-sensitive with drafts that might expire.

- The Internet Engineering Task Force
<http://www.ietf.org/>
- The C LDAP Application Program Interface Internet-Draft
<http://www.mozilla.org/directory/ietf-docs/draft-ietf-ldapext-ldap-c-api-05.txt>
- LDAP Control Extension for Server Side Sorting of Search Results, RFC 2891
<http://www.faqs.org/rfcs/rfc2891.html>
- Simple Authentication and Security Layer (SASL)
<http://www.faqs.org/rfcs/rfc2222.html>
- Secure Sockets Layer (SSL) Protocol (v3), Internet-Draft Revision 3
<http://wp.netscape.com/eng/ssl3/>
- Netscape Portable Runtime Library
<http://www.mozilla.org/projects/nspr>

Acknowledgements

This product includes software developed by the Apache Software Foundation:

<http://www.apache.org/>

This product also includes software developed through Mozilla

<http://www.mozilla.org/>

Documentation Feedback

Sun is interested in improving its documentation and welcomes your comments and suggestions. Use this web-based form to provide feedback to Sun:

<http://www.sun.com/hwdocs/feedback/>

Please provide the full document title and part number in the appropriate fields. The part number can be found on the title page of the book or at the top of the document, and is usually a seven or nine digit number. For example, the part number of this Directory Server Resource Kit 5.2.1 *LDAP SDK for C Programming Guide* is 817-6707-05.

Getting Started

Chapter 1, “Introduction”

Chapter 2, “Sample Client”

Introduction

The Sun™ Java System Directory Server Resource Kit (DSRK) *LDAP SDK for C Programming Guide* provides instructions for the Lightweight Directory Access Protocol (LDAP) software development kit (SDK) for C as well as usage information for the application programming interfaces (APIs). This introductory chapter contains the following sections:

- Overview
- Provided Directories and Files
- Compiling Applications
- The Example Programs

Overview

The DSRK includes the Sun Java System LDAP SDK for C. The LDAP SDK for C includes the C libraries for the LDAP API as well as sample code that demonstrates how to call most of the functions. The APIs are defined by the header files that declare all of the functions, data types and code values available in the binaries. You use the functions contained in this API to write C or C++ client applications that take full advantage of the capabilities of Sun Java System Directory Server. The API also includes custom extensions that give access to the latest features in Directory Server 5.2.

NOTE	Because they are built around core functions of the LDAP v2 and v3 standards, the API can be used to interact with any conforming LDAP server. This API conforms to the Internet Engineering Task Force (IETF) standard proposed in <i>The C LDAP Application Programming Interface</i> (http://www.ietf.org/proceedings/00dec/I-D/draft-ietf-ldapext-ldap-c-api-05.txt).

The LDAP API

The *Lightweight Directory Access Protocol (v3), RFC 2251*

(<http://www.faqs.org/rfcs/rfc2251.html>) defines a set of operations that must be used to access data in a LDAPv3-compliant directory server. The functionality implemented in the LDAP SDK for C closely follows these operations as a C API is defined for each operation. Using the C API, you can enable your clients to connect to LDAPv3-compliant directory servers and perform standard or extended LDAP functions such as:

- Search for and retrieve a list of entries.
- Add new entries to the database.
- Update existing entries.
- Delete entries.
- Rename entries.

For example, if you are writing an email application, you can use the functions in the API to retrieve email addresses from an LDAP server.

Synchronous and Asynchronous Operations

The API functions allow you to perform LDAP operations synchronously or asynchronously. (The only differences between these two options are in the calling convention; the LDAP exchanges are identical.)

- You can call a synchronous function if you want the client applications' calling thread to wait for the operation to complete before receiving the return value of a function.
- You can call an asynchronous function and poll the LDAP client library to check the status of the operation if you want to perform other work while waiting for an operation to complete.

For more information on synchronous and asynchronous operations, see “Synchronous Examples” on page 49 and “Asynchronous Examples” on page 50 as well as “Managing Memory” on page 86 of Chapter 4, “Using the LDAP API.” For a complete list of the functions implemented in this SDK, refer to Chapter 17, “Function Reference.”

Installation Instructions

Instructions on installing the DSRK can be found in the *Sun Java System Directory Server Resource Kit Tools Reference* (<http://docs.sun.com/doc/816-6400-10>). After installation, the base directory for the LDAP SDK for C on a Solaris™ operating system is *DSRK_base/lib/ldapcsdk*.

Provided Directories and Files

The DSRK, in addition to the utilities discussed in the *Sun Java System Directory Server Resource Kit Tools Reference* (<http://docs.sun.com/doc/816-6400-10>) and the SDK discussed in the *Sun Java System Directory Server Resource Kit LDAP SDK for Java Programming Guide* (<http://docs.sun.com/doc/817-6708-05>), installs C header files, C libraries, tools, and example programs for use with the LDAP SDK for C. These files are contained in the directories outlined in Table 1-1. The directories are installed in *DSRK_base/lib/ldapcsdk*.

Table 1-1 LDAP SDK for C Directories

Directory	Description
etc	This directory contains miscellaneous files that can be modified and used. They are described in this chapter's section, "Miscellaneous Files."
examples	This directory contains sample source code and makefiles for LDAP clients. See the README file in this directory for more information.
include	This directory contains the header files. You must include the files contained in this directory in your client source files. The header files are described in this chapter's section, "Header Files."
include-nspr	This directory contains Netscape Portable Runtime (NSPR) header files. The NSPR provides a platform-neutral API for system level and libc-like functions. More information on these headers and NSPR in general can be found in the <i>NSPR Reference</i> (http://www.mozilla.org/projects/nspr/reference/html/index.html).
include-private	This directory contains private header files that are not documented in this guide.
lib	This directory contains the C library files. The specific library used is dependent on the type of application you are building. For details, see this chapter's section, "Libraries."
lib-private	This directory contains private libraries that are not documented.

Table 1-1 LDAP SDK for C Directories

Directory	Description
tools	This directory contains the LDAP command-line tools. They are described in the <i>Tools Reference</i> (http://docs.sun.com/doc/816-6400-10). In order to use these applications, you need to ensure that they can find the LDAP API shared library or dynamic link library.

Header Files

Header files contain information about the library that programs using it need to know; for example, it might contain constants and types, along with prototypes for functions available in the library. The LDAP SDK for C header files can be found in the *DSRK_base/lib/ldapsdk/include* directory. They are listed in Table 1-2.

Table 1-2 LDAP SDK for C Header Files

Header File	Description
disptmpl.h	A header file related to the templates (ldaptemplates.conf).
iutil.h	Contains the interface for libiutil, the Innosoft migration library.
lber.h	Contains prototypes for the standard Basic Encoding Rules (BER) functions, structures and defines. More information is in the section on "lber.h."
ldap-deprecated.h	Contains deprecated functions that should not be used. This header is included by ldap.h.
ldap-extension.h	Contains functions, structures and defines that extend the standard LDAP C API specification. This header is included by ldap.h.
ldap-platform.h	A private header file that contains platform-specific definitions / abstraction layer.
ldap-standard.h	Contains the standard LDAP functions, structures and defines. This header is included in ldap.h.
ldap-to-be-deprecated.h	Contains functions, structures and defines that will be deprecated in future releases. This header is included in ldap.h.
ldap.h	This base header contains LDAP functions, structures, and defines to mirror the latest LDAP C API specifications. This header also includes the ldap-deprecated.h, ldap-extension.h, ldap-standard.h, and ldap-to-be-deprecated.h headers. More information is in "ldap.h."
ldap_ssl.h	Contains prototypes for LDAP over SSL functions, structures, and defines. More information is in "ldap_ssl.h."

Table 1-2 LDAP SDK for C Header Files (*Continued*)

Header File	Description
ldappr.h	Contains prototypes for the functions, structures, and defines contained in the Netscape Portable Runtime. More information is in "ldappr.h."
srchpref.h	A header file related to the search preferences (ldapsearchprefs.conf).

ldap.h

To make use of the functions contained in the LDAP SDK for C, whether you are developing on a UNIX®-based or 32-bit Microsoft® Windows system, you must include the `ldap.h` header file in your C source files as shown in the following line of code:

```
#include "ldap.h"
```

Including this file in your source also includes the following header files:

- `ldap-deprecated.h`
- `ldap-extension.h`
- `ldap-standard.h`
- `ldap-to-be-deprecated.h`

NOTE `ldap-platform.h` is also included via its inclusion in `ldap-standard.h`.

lber.h

`lber.h` is included in `ldap-standard.h`; therefore you do not need to explicitly include it. Basic Encoding Rules (BER) is a simple tag / value scheme that encodes requests and decodes results as well as sending and receiving said requests.

ldap_ssl.h

If you are calling the LDAP over SSL functions, you also need to include the `ldap_ssl.h` header file, as follows:

```
#include "ldap_ssl.h"
```

ldappr.h

To make use of the Netscape Portable Runtime (NSPR) with your LDAP applications, you must include `ldappr.h`. NSPR is a set of platform-neutral API that provides system functions such as threads, thread synchronization, I/O, interval timing, and atomic operations. This header file contains prototypes for functions that tie the LDAP libraries to NSPR. You include it as follows:

```
#include "ldappr.h"
```

You can find more information on NSPR in the `ldappr.h` header file itself and at the Netscape Portable Runtime project page. Additional information regarding the DSRK and NSPR can be found in “Dependencies.”

Miscellaneous Files

The LDAP SDK for C ships with the the sample files described in Table 1-3. They can be called by the end user specifically when using certain API. They are in the `DSRK_base/lib/ldapcsdk/etc` directory.

Table 1-3 Sample Configuration Files

File Name	Description
<code>ldapfilter.conf</code>	This LDAP filter configuration file can be used in context with <code>ldap_init_getfilter()</code> .
<code>ldapfriendly</code>	This file is used to map the two letter country codes to their full names by <code>ldap_friendly_name()</code> .
<code>ldapsearchprefs.conf</code>	This configuration file was used in context with a deprecated function.
<code>ldaptemplates.conf</code>	This configuration file was used in context with a deprecated function.

Libraries

The LDAP SDK for C comes with several different libraries. A library is a set of ready-made functions which are linked into a program. There are two types of libraries: *static* and *shared*. Static (or *archive*) libraries are compiled with the code. Shared libraries are dynamically loaded into memory when needed, reducing the size of the executable. The specific library files to which you link is dependent on the type of application you are building.

NOTE	Static libraries, although also installed, are not listed nor described.

Library Naming Conventions

Libraries on different systems have different naming conventions. Table 1-4 details the conventions relevant to Sun Java System DSRK.

Table 1-4 Library Naming Convention by Operating System

Operating System	Static Library Name	Shared Library Name
Solaris	<code>liblibraryname.a</code>	<code>liblibraryname.SO.versionnumber</code>
Windows	<code>nslibraryname.lib</code>	<code>nslibraryname.dll</code>
HP-UX	<code>liblibraryname.a</code>	<code>liblibraryname.Sl*</code>

On UNIX-based systems, the shared library file name may be fully qualified by prefixing path information.

Installed Shared Libraries

Table 1-5 outlines the installed libraries. They can be found in the `DSRK_base/lib/ldapcsdk/lib` directory.

Table 1-5 Shared Libraries

UNIX-based Library	Windows Library	Description
<code>libldap50.so</code>	<code>nsldap32v50.dll</code>	LDAP library
<code>libprldap50.so</code>	<code>nsldappr32v50.dll</code>	LDAP library built with NSPR. This library requires the <code>libnspr4.so</code> library installed in the <code>DSRK_base/lib/ldapcsdk/lib-private</code> directory.
<code>libssldap50.so</code>	<code>nsldapssl32v50.dll</code>	LDAP library built with support for Secure Sockets Layer. It requires the <code>libnss3.so</code> and <code>libnspr4.so</code> libraries installed in the <code>DSRK_base/lib/ldapcsdk/lib-private</code> directory.

Dependencies

The following dependencies are essential when compiling your application.

- Libraries built with NSPR require a link to the `libnspr4.so` library installed in `DSRK_base/lib/ldapcsdk/lib-private`. `libnspr4.so` contains the core functions of the NSPR.

- The libraries built with support for Secure Sockets Layer (SSL) require a link to the `libnss3.so` and `libnspr4.so` libraries installed in `DSRK_base/lib/ldapcsdk/lib-private`. `libnspr4.so` contains the core functions of the NSPR and `libnss3.so` contains the Netscape Security Services (NSS) which provides encryption, cryptographic and SSL / Public Key Infrastructure (PKI) support.

LDAP Tools

The LDAP SDK for C includes several utilities to help you work with LDAP data sets. They are installed in the `DSRK_base/lib/ldapcsdk/tools` directory (and duplicated in the `DSRK_base/bin/dsrk52` directory). Table 1-6 outlines the tools; more information can be found in the *Tools Reference* (<http://docs.sun.com/doc/816-6400-10>). A list of options can be retrieved by typing the tool name at the command line.

Table 1-6 LDAP Utilities

Utility	Description
<code>ldapcmp</code>	Compares the contents of a single LDAP entry or subtree in two directories.
<code>ldapcompare</code>	Compares an attribute value against the contents of a given LDAP entry.
<code>ldapdelete</code>	Deletes existing LDAP entries.
<code>ldapmodify</code>	Edits the contents of a LDAP directory, either by adding new entries or modifying existing ones.
<code>ldapsearch</code>	Issues search requests to an LDAP directory and displays the result as LDAP Data Interchange Format (LDIF) text.

Compiling Applications

As detailed in “Provided Directories and Files,” the DSRK includes header files and corresponding shared libraries for its API functions. You must include the header files and link to the libraries when compiling your application. Inclusion of the header files is discussed in “Header Files.” Linking to shared libraries is done when compiling the application; this is discussed in the following sections.

NOTE	The LDAP SDK for C is tested and certified to work with C compilers. It is not guaranteed to work with C++ compilers.

Compiling on UNIX Platforms

When compiling clients on UNIX platforms, you might need to specify a link option to link the application to the appropriate shared libraries. When compiling the application (using the `gcc` or `cc` command), you would use the `-L` option to add a new path and file for which the link editor can search. (Path names defined using the `-L` option are used only by the link editor implementation of the compiler application.) For example, on Solaris, you might use the following link option:

```
gcc ... -Lpath_to_file -file_name ...
```

Where:

- *path_to_file* is the path to the file to be linked.
- *file_name* is the name of the file to be linked.

NOTE

When running clients built with shared object libraries, make sure to set the appropriate environment variable that specifies the runtime path used to find libraries. For example, on Solaris and Linux, set the `LD_LIBRARY_PATH` environment variable to the location of the shared library. On HP-UX, set the `SHLIB_PATH` environment variable if you are using the `-WL, +S+b` flag to compile your client application. On AIX, set the `LIBPATH` environment variable.

Refer to `Makefile` in the `DSRK_base/lib/ldapcsdk/examples` directory for more details on compiling your applications on UNIX platforms.

Compiling on Windows Systems

The DSRK provides both static (or *import*) and dynamic versions of its Windows-based libraries. When compiling clients on 32-bit Windows systems (Windows 2000, Windows 2003, Windows XP, Windows NT, Windows 95, Windows 98, and Windows ME), make sure to link to the SSL-enabled static library. The name of the library, installed in `DSRK_base/WindowsVersion/lib/ldapcsdk/lib`, is `nsldapssl32v50.lib`.

NOTE

The LDAP SDK for C does not support 16 or 64-bit Windows systems.

Using the DLL

You should also copy the `nsldap32v50.dll` dynamic link library (DLL) to a directory where your client can find it. Sun Java System Directory Server might install a different version of the `nsldap32v50.dll` in the Windows `system32` directory. Make sure that your client finds the version of the DLL that is included with the DSRK before it finds the version in the `system32` directory. For example, you can copy the DSRK DLL to the same directory as your client application. During run-time, your client searches for the DLL in the following locations:

1. The directory from which the application loaded.
2. The current directory.
3. The Windows system directory, typically `winnt\system32`.

To avoid potential conflicts, do not install the DSRK DLL in this directory.

4. The directories listed in the `PATH` environment variable.

Definitions

A Windows application can use either a command-line interface, or a graphical user interface (GUI). Under 32-bit Windows systems, there is also the notion of a console application in which all user interaction happens via a standard DOS shell interface, with a `FILE` pointer like that in Unix. This does not exist under Win16, where a Windows window is required. Because of this, make sure to define:

- `_CONSOLE` if you are writing a console application.
- `_WINDOWS` if you are writing a standard Windows GUI application.

Linking To NSPR Libraries

The NSPR libraries (as discussed in “`ldappr.h`”) are distributed with the LDAP SDK for C. When linking with SSL-enabled applications, you must also link with the NSPR libraries. For more information on the NSPR libraries, see the *NSPR Reference* (<http://www.mozilla.org/projects/nspr/reference/html/index.html>).

Linking Dependencies

When you run LDAP clients on UNIX-based or Windows system, you must ensure that the operating system can find the shared libraries that support the functions called by your application. (Generally, these files are referred to as *runtime* libraries.) Any of the following options will accomplish this.

- Make sure that the shared library file (such as `libldap50.so` for Solaris) is in a location specified by environment variables.
- On some platforms, if you have compiled your client with certain flags set, you can set an environment variable to specify the runtime path to check when loading libraries. For example, on Solaris and Linux, you can use the `LD_LIBRARY_PATH` environment variable if you use the `-Wl, +s+b` flag when compiling and linking. (On HP-UX, you can use the `SHLIB_PATH` environment variable, and on AIX, you can use the `LIBPATH` environment variable.)
- Use a link flag that specifies the path where the executable can find the library. For example, on Solaris, you can use the `-R` flag to specify the path where the executable can find the library.

See the `Makefile` in the `examples` directory for examples of additional flags and defines that you might need to specify when compiling and linking your LDAP client. Different platforms might require different sets of define statements.

The Example Programs

The DSRK ships with several examples that demonstrate the use of the functions contained in the LDAP SDK for C library. Located in the `DSRK_base/lib/ldapcsdk/examples` directory, the example code assumes that you are running clients against the LDAPv3-compliant Sun Java System Directory Server and that you have the sample database properly loaded. For details on the source files, refer to the `README` contained in the `examples` directory.

TIP	The samples use synchronous LDAP calls and their asynchronous counterparts. The synchronous LDAP calls are more straightforward than their asynchronous counterparts therefore, it is suggested you look at these examples first.

Synchronous Examples

The synchronous calls block the calling process until all results have been returned. Because these programs usually rely on event loops, they are not appropriate for use with clients that implement a GUI in a single-threaded environment. However, these sample programs work fine with command-line clients and CGI programs. Table 1-7 lists the synchronous examples in the LDAP SDK for C.

Table 1-7 Synchronous Example Programs

Example Source	Description
search.c	Shows how to use <code>ldap_search_s()</code> to search for all entries which have an attribute value that exactly matches what you're searching for. In this example, all entries with the surname (last name) <i>Jensen</i> are retrieved and displayed.
csearch.c	Like <code>search.c</code> , but enables an in-memory cache.
ssnoauth.c	Like <code>search.c</code> , but the search is done over an SSL-protected TCP connection.
ssearch.c	Like <code>ssnoauth.c</code> , but with certificate-based authentication thrown in.
srvrsort.c	Shows how to use server-side sorting in conjunction with the <code>ldap_search_ext_s()</code> function.
rdentry.c	Shows how to use <code>ldap_search_s()</code> to retrieve a particular entry from the directory.
getattrs.c	Retrieves specific attributes from an entry.
compare.c	Shows how to use <code>ldap_compare_s()</code> , which allows you to test if a particular value is contained in an attribute of an entry.
modattrs.c	Shows how to use <code>ldap_modify_s()</code> to replace and add to values in an attribute.
modrdn.c	Shows how to use <code>ldap_modrdn2_s()</code> to change the relative distinguished name (RDN) of an entry.
getfilt.c	Shows how to use the <code>ldap_getfilter*()</code> family of functions, which help generate LDAP filters based on an arbitrary search string provided by a user.
crtfilt.c	Shows how to use the <code>ldap_create_filter()</code> function to generate LDAP filters.

Asynchronous Examples

These examples use the asynchronous LDAP calls; you begin an operation, and periodically poll to see if any results have been returned. Table 1-8 lists the asynchronous examples in the LDAP SDK for C.

Table 1-8 Asynchronous Example Programs

Example Source	Description
asearch.c	Initiates a search for entries, printing the results as they arrive.

Table 1-8 Asynchronous Example Programs (*Continued*)

Example Source	Description
nsprio.c	Like asearch.c but uses the PerLDAP routines to incorporate Netscape Portable Runtime (NSPR).
add.c	Adds an entry to the directory.
del.c	Deletes an entry from the directory.
psearch.c	Shows how to use Persistent Search (an LDAPv3 extension) to monitor a directory server for changes.
ppolicy.c	Attempts to bind to the directory and reports back any password expiration information received. This demonstrates how clients can process password policy information.

Sample Client

The Sun™ Java System LDAP SDK for C provides a simple example of a Lightweight Directory Access Protocol (LDAP) client. This chapter explains the details. It contains the following sections:

- The Source Code
- Compiling the Client
- Running the Client

The Source Code

Code Example 2-1 is the source code for a command-line program that retrieves the full name, last name, email address, and telephone number of *Barbara Jensen*.

Code Example 2-1 Source Code to Retrieve LDAP Entry

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

/* Adjust these setting for your own LDAP server */
#define HOSTNAME "localhost"
#define PORT_NUMBER LDAP_PORT
#define FIND_DN "uid=bjensen,ou=People,dc=example,dc=com"

int
main( int argc, char **argv )
{
    LDAP          *ld;
    LDAPMessage    *result, *e;
    BerElement     *ber;
    char           *a;
    char           **vals;
    int            i, rc;
```

Code Example 2-1 Source Code to Retrieve LDAP Entry (*Continued*)

```

/* Get a handle to an LDAP connection. */
if ( (ld = ldap_init( HOSTNAME, PORT_NUMBER )) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
}

/* Bind anonymously to the LDAP server. */
rc = ldap_simple_bind_s( ld, NULL, NULL );
if ( rc != LDAP_SUCCESS ) {
    fprintf(stderr, "ldap_simple_bind_s: %s\n", ldap_err2string(rc));
    return( 1 );
}

/* Search for the entry. */
if ( ( rc = ldap_search_ext_s( ld, FIND_DN, LDAP_SCOPE_BASE,
    "(objectclass=*)", NULL, 0, NULL, NULL, LDAP_NO_LIMIT,
    LDAP_NO_LIMIT, &result ) ) != LDAP_SUCCESS ) {
    fprintf(stderr, "ldap_search_ext_s: %s\n", ldap_err2string(rc));
    return( 1 );
}

/* Since we are doing a base search, there should be only
   one matching entry. */
e = ldap_first_entry( ld, result );
if ( e != NULL ) {
    printf( "\nFound %s:\n\n", FIND_DN );

    /* Iterate through each attribute in the entry. */
    for ( a = ldap_first_attribute( ld, e, &ber );
        a != NULL; a = ldap_next_attribute( ld, e, ber ) ) {

        /* For each attribute, print the attribute name and values. */
        if ((vals = ldap_get_values( ld, e, a)) != NULL ) {
            for ( i = 0; vals[i] != NULL; i++ ) {
                printf( "%s: %s\n", a, vals[i] );
            }
            ldap_value_free( vals );
        }
        ldap_memfree( a );
    }
    if ( ber != NULL ) {
        ber_free( ber, 0 );
    }
}
ldap_msgfree( result );
ldap_unbind( ld );
return( 0 );
}

```

Compiling the Client

The method used to compile the source code depends on the operating system on which you will run the application. The following sections include instructions for compiling on the Solaris™ Operating System and Microsoft® Windows NT.

Compiling on Solaris

The examples directory (located in *DSRK_base/lib/ldapcsdk*) contains a UNIX® Makefile. You can modify the Makefile to compile the sample by adjusting the flags specified in the file. The Makefile assumes that the LDAP SDK for C header files are located in the *../include* directory. Code Example 2-2 details the Makefile flags.

Code Example 2-2 Solaris Makefile for Sample Client

```
#
# Makefile for the example in this chapter.
#
EXTRACFLAGS=
EXTRALDFLAGS=-lsocket -lnsl
LDAPLIB=ldap41 -lplc3 -lplds3 -lnspr3
INCDIR=../include
LIBDIR=../lib
LIBS=-L$(LIBDIR) $(LDAPLIB) $(EXTRALDFLAGS)
OPTFLAGS=-g
CFLAGS=$(OPTFLAGS) -I$(INCDIR) $(EXTRACFLAGS)
CC=cc
PROGS=rdentry
all:          $(PROGS)
rdentry:      rdentry.o
              $(CC) -o rdentry rdentry.o $(LIBS)
clean:
              /bin/rm -f $(PROGS) *.o a.out core
```

Compiling on Windows NT

If you are compiling the sample client on Windows NT, set up a makefile for this application. Make sure to:

- Create a new project workspace for a console application and add the source file to the project if you are using Microsoft Visual C++.

- Set your options to include `lib` as one of the directories containing library files and `include` as one of the directories containing include files.
- Link to `nsldap32v41.lib`, the LDAP API import library for Windows.

Running the Client

Before running the sample client, make sure that your LDAP server is set up with the entry the sample will attempt to find. Unless you change the source code in Code Example 2-1 on page 53, the entry would be for the full name, last name, email address, and telephone number of *Barbara Jensen*.

Running on UNIX

If you have compiled and linked the client on a UNIX platform, make sure to set your `LD_LIBRARY_PATH` environment variable to the location of the `libldap41.so`, `plc3`, `plds3`, and `nspr3` library files.

NOTE	As an alternative, when linking the file, specify the option that identifies the library directories for which the runtime linker should search. For example, on Solaris, use the <code>-R</code> to specify the location of the <code>libldap41.so</code> file.

Running on Windows NT

If you have linked the client with the `nsldap32v41.lib` import library on Windows NT, make sure to copy the `nsldap32v41.dll` file to one of the following directories:

- The directory from which the application loaded.
- The current directory.
- The 32-bit Windows system directory. In Windows NT, this directory is typically `winnt\system32`.
- The Windows directory.
- The directories listed in the `PATH` environment variable.

Working with the LDAP SDK for C

- Chapter 3, “Writing an LDAP Client”
- Chapter 4, “Using the LDAP API”
- Chapter 5, “Searching a LDAP Directory”
- Chapter 6, “Using Filter Configuration Files”
- Chapter 7, “Adding, Modifying and Deleting Entries”
- Chapter 8, “Comparing Entries”
- Chapter 9, “Working with LDAP URLs”

Writing an LDAP Client

With the Sun™ Java System LDAP SDK for C, you can write a new application, or enable an existing application, to interact with an Lightweight Directory Access Protocol (LDAP) server. This chapter covers the procedures for connecting to an LDAP server, authenticating, requesting operations, and disconnecting from the server. It includes the following sections:

- Designing an LDAP Client
- Initializing an LDAP Session
- Setting Session Preferences
- Binding and Authenticating to an LDAP Server
- Performing LDAP Operations
- Closing the Connection to the Server

Designing an LDAP Client

The following procedure outlines a typical process for communicating with an LDAP server.

1. Initialize an LDAP session.

See “Initializing an LDAP Session” on page 61 for details.

2. Bind to the LDAP server, if necessary.

See “Binding and Authenticating to an LDAP Server” on page 66 for details.

3. Perform LDAP operations, such as searching the directory or modifying entries in the directory.

See “Performing LDAP Operations” on page 73 for details.

4. Close the connection to the LDAP server when finished.

See “Closing the Connection to the Server” on page 73 for details.

Code Example 3-1 is source code for a client that requests a LDAP search operation from a server. The client connects to the LDAP server (running on the local machine at port 389), searches the directory for entries with the last name *Jensen* (sn=Jensen), and prints out the distinguished name (DN) of any matching entry.

Code Example 3-1 Source Code for LDAP search operation

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

/* Specify the search criteria here. */
#define HOSTNAME "localhost"
#define PORTNUMBER 389
#define BASEDN "dc=example,dc=com"
#define SCOPE LDAP_SCOPE_SUBTREE
#define FILTER "(sn=Jensen)"

int
main( int argc, char **argv )
{
    LDAP          *ld;
    LDAPMessage    *result, *e;
    char          *dn;
    int            version, rc;
    /* Print out an informational message. */
    printf( "Connecting to host %s at port %d...\n\n", HOSTNAME,
        PORTNUMBER );

    /* STEP 1: Get a handle to an LDAP connection and
       set any session preferences. */
    if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
        perror( "ldap_init" );
        return( 1 );
    }

    /* Use the LDAP_OPT_PROTOCOL_VERSION session preference to specify
       that the client is an LDAPv3 client. */
    version = LDAP_VERSION3;
    ldap_set_option( ld, LDAP_OPT_PROTOCOL_VERSION, &version );

    /* STEP 2: Bind to the server.
       In this example, the client binds anonymously to the server
       (no DN or credentials are specified). */
    rc = ldap_simple_bind_s( ld, NULL, NULL );
    if ( rc != LDAP_SUCCESS ) {
```

Code Example 3-1 Source Code for LDAP search operation (*Continued*)

```

        fprintf(stderr, "ldap_simple_bind_s: %s\n", ldap_err2string(rc));
        return( 1 );
    }

    /* Print out an informational message. */
    printf( "Searching the directory for entries\n"
           " starting from the base DN %s\n"
           " within the scope %d\n"
           " matching the search filter %s...\n\n",
           BASEDN, SCOPE, FILTER );

    /* STEP 3: Perform the LDAP operations.
       In this example, a simple search operation is performed.
       The client iterates through each of the entries returned and
       prints out the DN of each entry. */
    rc = ldap_search_ext_s( ld, BASEDN, SCOPE, FILTER, NULL, 0,
                           NULL, NULL, NULL, 0, &result );
    if ( rc != LDAP_SUCCESS ) {
        fprintf(stderr, "ldap_search_ext_s: %s\n", ldap_err2string(rc));
        return( 1 );
    }
    for ( e = ldap_first_entry( ld, result ); e != NULL;
          e = ldap_next_entry( ld, e ) ) {
        if ( (dn = ldap_get_dn( ld, e )) != NULL ) {
            printf( "dn: %s\n", dn );
            ldap_memfree( dn );
        }
    }
    ldap_msgfree( result );

    /* STEP 4: Disconnect from the server. */
    ldap_unbind( ld );
    return( 0 );
}
...

```

Initializing an LDAP Session

Before connecting to an LDAP server, you need to initialize a session. As part of this process, you create an LDAP structure that contains information about the LDAP server and session. You then pass this LDAP structure (usually as a pointer) to all subsequent functions in order to identify the LDAP server with which you are working. Sample code for initializing an LDAP session is provided in the section, “Session Initialization Sample Code” on page 63.

NOTE If you plan to connect to the LDAP server over a Secure Sockets Layer (SSL), the procedure for initializing an LDAP session is different. For details, see Chapter 11, “Connecting Over SSL.”

Specifying a Single LDAP Server

To initialize a single LDAP session, call `ldap_init()` and pass the host name and port number of your LDAP server as arguments to the function. If the server is using the default port 389 for the LDAP, you can pass `LDAP_PORT` as the value for the `defport` parameter as illustrated in Code Example 3-2.

Code Example 3-2 Passing an LDAP Server with the Default LDAP Port

```
...
LDAP *ld
...
ld = ldap_init( "directory.iplanet.com", LDAP_PORT );
```

If successful, `ldap_init()` returns a *connection handle* (a pointer to the LDAP structure that contains information about the connection) to the LDAP server. You need to pass this pointer to the API for connecting, authenticating, and performing LDAP operations on a server. For example, when you call a function to search the directory, you need to pass the connection handle as a parameter to provide a context for the connection.

CAUTION `ldap_init()` does not open a connection to the LDAP server. The actual connection opening will occur when the first operation is attempted.

Specifying a List of LDAP Servers

When initializing the LDAP session, you can also specify a list of LDAP servers for which you want to attempt connections. If the first LDAP server in the list does not respond, the client will attempt to connect to the next server in the list. To specify a list of LDAP servers, pass a space-delimited list of the host names as the first argument to the `ldap_init()` function. In Code Example 3-3, the LDAP client will

attempt to connect to the LDAP server on `ld1.example.com`, port 389. If that server does not respond, the client will attempt to connect to the LDAP server on `ld2.example2.com`, port 389. If that server does not respond, the client will use the server on `ld3.example.com`, port 389.

Code Example 3-3 Passing Multiple LDAP Servers with Default LDAP Port

```
...
LDAP *ld
...
ld = ldap_init( "ld1.example.com ld2.example2.com
               ld3.example.com", LDAP_PORT );
```

If any of the servers do not use the default LDAP port, use the *host:port* format to specify the server name and port number as the argument to `ldap_init()`. In Code Example 3-4, the client will attempt to connect to the LDAP server on `ld1.example.com`, port 389. If that server does not respond, the client will attempt to connect to the LDAP server on `ld2.example.com`, port 38900.

Code Example 3-4 Passing Non-Default LDAP Ports

```
...
LDAP *ld
...
ld = ldap_init( "ld1.example.com ld2.example.com:38900",
               LDAP_PORT );
```

Session Initialization Sample Code

Code Example 3-5 initializes a session with an LDAP server, specifying a list of LDAP servers to try: `ldap.example.com:389` and `directory.example.com:3890`. It also sets a session preference that identifies the client as an LDAPv3 client. (After you initialize a session with an LDAP server, you can set session preferences. More information can be found in the section, “Setting Session Preferences” on page 64.)

NOTE Note that `ldap_init()` does not connect to the LDAP server right away.

Code Example 3-5 Initializing an LDAP Session

```

#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
int ldap_default_port, version;

/* Specify list of LDAP servers that you want to try connecting to. */
char *ldap_host = "ldap.example.com directory.example.com:3890";

/* If the LDAP server is running on the standard LDAP port (port 389),
 * you can use LDAP_PORT to identify the port number. */
ldap_default_port = LDAP_PORT;
...
/* Initialize the session with the LDAP servers. */
if ( ( ld = ldap_init( ldap_host, ldap_default_port ) ) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
}

/* Specify the LDAP version supported by the client. */
version = LDAP_VERSION3;
ldap_set_option( ld, LDAP_OPT_PROTOCOL_VERSION, &version );

...
/* Subsequent API calls pass ld as argument to identify the LDAP server. */
...

```

Setting Session Preferences

With the LDAP SDK for C, you can set preferences for your client that you want applied to all LDAP sessions. To get or set the value of a preference, call either the `ldap_get_option()` or `ldap_set_option()` functions, respectively. Both functions pass two parameters (in addition to the `ld` parameter which represents the connection to the server):

- The **option** parameter identifies the option that you want to get or set.
- The **value** parameter is either a pointer to a place to put the value that you are getting or a pointer to the value that you are setting.

TIP You can set a preference for all connections created by passing `NULL` (rather than an LDAP structure which specifies the connection) as the first argument.

The following sections illustrate how to set two specific preferences. For a complete listing of the options and values you can get and set, see the documentation for the two functions in Chapter 17, “Function Reference.”

Setting the Restart Preference

If communication with the LDAP server is interrupted, the result code `LDAP_SERVER_DOWN` is returned by your client. If you want your client to continue to attempt communication with the server, you can set the `LDAP_OPT_RECONNECT` preference for the session. Once set, if your connection is lost, the client will attempt another bind (using the same authentication) to reestablish the connection.

To set the restart preference, call the `ldap_set_option()` function and pass `LDAP_OPT_RECONNECT` as the value of the `option` parameter. If you want the client to resume LDAP I/O operations automatically, set the `optdata` parameter to `LDAP_OPT_ON` to specify that the same connection handle can be used to reconnect to the server. If you do not want the client to resume I/O operations, set the `optdata` parameter to `LDAP_OPT_OFF` to specify that you want to create a new connection handle to connect to the server. By default, the `optdata` parameter is set to `LDAP_OPT_OFF`. Both `LDAP_OPT_OFF` and `LDAP_OPT_ON` are cast to `(void *)`. Code Example 3-6 illustrates how you can pass these parameters directly to the function.

Code Example 3-6 Passing Restart Preferences to Function

```
...
ldap_set_option( ld, LDAP_OPT_RECONNECT, LDAP_OPT_ON );
...
```

Specifying the LDAP Version of Your Client

If you plan to call functions that make use of LDAPv3 features (such as controls or extended operations), you should set the protocol version of your client to LDAPv3. By default, clients built with the Sun Java System LDAP SDK for C identify themselves to LDAP servers as LDAPv2 clients.

To specify the LDAP version supported by your client, call the `ldap_set_option()` function and set the `LDAP_OPT_PROTOCOL_VERSION` option to the value 3. Code Example 3-7 illustrates this.

Code Example 3-7 Passing the LDAP Protocol Version Number

```
...  
version = LDAP_VERSION3;  
ldap_set_option( ld, LDAP_OPT_PROTOCOL_VERSION, &version );  
...
```

After setting this option, as part of the authentication process, your client sends the supported LDAP version number to the server allowing it to determine whether or not to enable LDAPv3 features.

TIP The LDAPv3 allows you to perform LDAP operations without first binding to the server. An LDAPv3 server will assume that the client is LDAPv3 compliant if the client issues non-bind operations before it issues a bind.

Binding and Authenticating to an LDAP Server

When connecting to the LDAP server, your client may need to send a bind request. A bind request should be sent if:

- You want to authenticate to the server to add or modify entries in a directory that requires authentication as a user with certain access privileges.
- You are connecting to a LDAPv2 server. LDAPv2 servers typically require clients to bind before any operations can be performed.

NOTE LDAP clients can also bind anonymously to the LDAP server if, for example, it doesn't require authentication for a simple directory search.

The bind request should contain the following information:

- The LDAP version of the client.
- The method of authentication that should be used.
- The DN that the client is attempting to authenticate as.
- The credentials to be used for authentication.

Methods of Authentication

When binding to an LDAP server, you can use a number of different methods to authenticate your client. They include:

Simple Authentication

With simple authentication, your client provides the DN of the user and the user's password to the LDAP server. For more information, see “Using Simple Authentication” on page 67.

NOTE	You can also use this method to bind anonymously by providing NULL values as the user's DN and password as described in “Binding Anonymously” on page 72.

Certificate-based Client Authentication

With certificate-based client authentication, your client sends its certificate to the LDAP server. For more information, see Chapter 11, “Connecting Over SSL.”

Simple Authentication and Security Layer (SASL)

SASL is detailed in the *Simple Authentication and Security Layer (SASL) - RFC 2222* (<http://www.faqs.org/rfcs/rfc2222.html>). Some LDAPv3 servers (including Sun Java System Directory Server) support authentication through SASL. For more information, see Chapter 12, “Using SASL Authentication.”

NOTE	If you are binding to Sun Java System Directory Server, the server may send back special controls to indicate that your password has expired or will expire in the near future. For more information on these controls, see “Using Password Policy Controls” in Chapter 13, “Working with LDAP Controls.”

Using Simple Authentication

If you plan to use simple authentication, call one of the following functions:

- `ldap_simple_bind_s()` is a synchronous function for use if you want to wait for the bind operation to complete before the function returns. See “Performing a Synchronous Authentication Operation” on page 68 for more information.

- `ldap_simple_bind()` is an asynchronous function for use if you do not want to wait for the bind operation to complete. With this function, you can perform other work while periodically checking for the results of the bind operation. See “Performing an Asynchronous Authentication Operation” on page 69 for more information.

NOTE For more information on the difference between synchronous and asynchronous functions, see “Synchronous and Asynchronous Functions” on page 75 in Chapter 4, “Using the LDAP API.”

Performing a Synchronous Authentication Operation

If you want to wait for the bind operation to complete before continuing, call `ldap_simple_bind_s()`. This function returns `LDAP_SUCCESS` if the operation completed successfully or an LDAP result code if a problem occurred. (See `ldap_simple_bind_s()` in Chapter 17, “Function Reference” for a list of result codes returned.)

NOTE If you specify a DN but no password, your client will bind to the server anonymously. If you want a NULL password to be rejected as incorrect, you need to write code to perform the check before you call `ldap_simple_bind_s()`.

Code Example 3-8 uses the synchronous `ldap_simple_bind_s()` function to authenticate user *Barbara Jensen* to the LDAP server.

Code Example 3-8 Synchronous Authentication

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

/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
#define BIND_DN "uid=bjensen,ou=People,dc=example,dc=com"
#define BIND_PW "hifalutin"

LDAP      *ld;
int        rc;
/* Get a handle to an LDAP connection. */
if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
}

/* Print out an informational message. */
printf( "Binding to server %s:%d\n", HOSTNAME, PORTNUMBER );
```

Code Example 3-8 Synchronous Authentication (*Continued*)

```

printf( "as the DN %s ...\n", BIND_DN );

/* Bind to the LDAP server. */
rc = ldap_simple_bind_s( ld, BIND_DN, BIND_PW );
if ( rc != LDAP_SUCCESS ) {
    fprintf(stderr, "ldap_simple_bind_s: %s\n\n", ldap_err2string(rc));
    return( 1 );
} else {
    printf( "Bind operation successful.\n" );
}

...
/* If you want, you can perform LDAP operations here. */
...

/* Disconnect from the server when done. */
ldap_unbind( ld );
return( 0 );
...

```

Performing an Asynchronous Authentication Operation

If you want to perform other work (in parallel) while waiting for the bind operation to complete, call `ldap_simple_bind()`. This function sends a LDAP bind request to the server and returns a message ID identifying the bind operation. You can check to see if your client has received the results of the bind operation from the server by calling the `ldap_result()` function and passing it this message ID. If your client has received the results, `ldap_result()` passes back the information in an `LDAPMessage` structure. To retrieve error information from `LDAPMessage`, you can pass it to the `ldap_parse_result()` function. `ldap_parse_result()` gets the LDAP result code of the operation and any error messages sent back from the server. This function also retrieves any controls sent back.

NOTE

If you specify a DN but no password, your client will bind to the server anonymously. If you want a NULL password to be rejected as incorrect, you need to write code to perform the check before you call `ldap_simple_bind()`.

Code Example 3-9 uses the asynchronous `ldap_simple_bind()` function to authenticate user *Barbara Jensen* to the LDAP server.

Code Example 3-9 Asynchronous Authentication

```

#include <stdio.h>
#include "ldap.h"

void do_other_work();
int global_counter = 0;
...

#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
#define BIND_DN "uid=bjensen,ou=People,dc=example,dc=com"
#define BIND_PW "hifalutin"

...
LDAP          *ld;
LDAPMessage   *res;
int           msgid = 0, rc = 0, parse_rc = 0, finished = 0;
char          *matched_msg = NULL, *error_msg = NULL;
char          **referrals;
LDAPControl   **serverctrls;
struct timeval zerotime;

/* Specify the timeout period for ldap_result(),
   which specifies how long the function should block when waiting
   for results from the server. */
zerotime.tv_sec = zerotime.tv_usec = 0L;

/* Get a handle to an LDAP connection. */
if ( ( ld = ldap_init( HOSTNAME, PORTNUMBER ) ) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
}

/* Print out an informational message. */
printf( "Binding to server %s:%d\n", HOSTNAME, PORTNUMBER );
printf( "as the DN %s ...\n", BIND_DN );

/* Send an LDAP bind request to the server. */
msgid = ldap_simple_bind( ld, BIND_DN, BIND_PW );

/* If the returned message ID is less than zero, an error occurred. */
if ( msgid < 0 ) {
    rc = ldap_get_lderrno( ld, NULL, NULL );
    fprintf(stderr, "ldap_simple_bind : %s\n", ldap_err2string(rc));
    ldap_unbind( ld );
    return( 1 );
}

/* Check to see if the bind operation completed. */
while ( !finished ) {
    rc = ldap_result( ld, msgid, 0, &zerotime, &res );
    switch ( rc ) {
        /* If ldap_result() returns -1, error occurred. */
        case -1:
            rc = ldap_get_lderrno( ld, NULL, NULL );

```

Code Example 3-9 Asynchronous Authentication (*Continued*)

```

fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
ldap_unbind( ld );
return ( 1 );

/* If ldap_result() returns 0, the timeout (specified by the
   timeout argument) has been exceeded before the client received
   the results from the server. Continue calling ldap_result()
   to poll for results from the server. */
case 0:
    break;

default:
    /* The client has received the result of the bind operation. */
    finished = 1;

    /* Parse this result to determine if the operation was successful.
       Note that a non-zero value is passed as the last parameter,
       which indicates that the LDAPMessage structure res should be
       freed when done. (No need to call ldap_msgfree().) */
    parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg,
        &error_msg, &referrals, &serverctrls, 1 );
    if ( parse_rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_parse_result: %s\n",
            ldap_err2string( parse_rc ) );
        ldap_unbind( ld );
        return( 1 );
    }
    /* Check the results of the operation. */
    if ( rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_simple_bind: %s\n",
            ldap_err2string( rc ) );

        /* If the server sent an additional error message,
           print it out. */
        if ( error_msg != NULL && *error_msg != '\0' ) {
            fprintf( stderr, "%s\n", error_msg );
        }

        /* If an entry specified by a DN could not be found,
           the server may also return the portion of the DN
           that identifies an existing entry.
           (See "Receiving the Portion of the DN Matching an Entry"
           for an explanation.) */
        if ( matched_msg != NULL && *matched_msg != '\0' ) {
            fprintf( stderr,
                "Part of the DN that matches an existing entry: %s\n",
                matched_msg );
        }
        ldap_unbind( ld );
        return( 1 );
    }
    else {
        printf( "Bind operation successful.\n" );
        printf( "Counted to %d while waiting for bind op.\n",
            global_counter );
    }
}

```

Code Example 3-9 Asynchronous Authentication (*Continued*)

```

        break;
    }
    /* Do other work here while waiting for results from the server. */
    if ( !finished ) {
        do_other_work();
    }
}

...
/* If you want, you can perform LDAP operations here. */
...

/* Disconnect from the server when done. */
ldap_unbind( ld );
return( 0 );
...
/* Function that does work while waiting for results from the server. */
void do_other_work() {
    global_counter++;
}
...

```

Binding Anonymously

In some cases, you may not need to authenticate to the LDAP server. For example, if you are writing a client to search the directory and users don't need special access permissions for the operation, you don't need to authenticate before performing the search. To bind as an anonymous user, call `ldap_simple_bind()` or `ldap_simple_bind_s()`, and pass `NULL` values for the `who` and `passwd` parameters as illustrated in Code Example 3-10.

Code Example 3-10 Binding Anonymously

```

...
rc = ldap_simple_bind_s( ld, NULL, NULL );
...

```

With LDAPv2, the client is required to send a bind request, even when binding anonymously (without specifying a name or password). With LDAPv3, the client is no longer required to bind to the server if the client does not need to authenticate.

Performing LDAP Operations

Once you initialize a session with an LDAP server and complete the authentication process, you can perform LDAP operations, such as searching the directory, adding new entries, updating existing entries, and removing entries (provided the server's access control allows these operations). Table 3-1 lists some of these LDAP operations and where you can get more information about them.

Table 3-1 Performing LDAP Operations

To Perform This Operation	Call This API Function	For More Information
Search for entries	<code>ldap_search_ext()</code> <code>ldap_search_ext_s()</code>	Chapter 5, "Searching a LDAP Directory"
Determine an attribute's value	<code>ldap_compare_ext()</code> <code>ldap_compare_ext_s()</code>	Chapter 8, "Comparing Entries"
Add entries	<code>ldap_add_ext()</code> <code>ldap_add_ext_s()</code>	Chapter 7, "Adding, Modifying and Deleting Entries"
Modify entries	<code>ldap_modify_ext()</code> <code>ldap_modify_ext_s()</code>	Chapter 7, "Adding, Modifying and Deleting Entries"
Delete entries	<code>ldap_delete_ext()</code> <code>ldap_delete_ext_s()</code>	Chapter 7, "Adding, Modifying and Deleting Entries"
Change DN of entries	<code>ldap_rename()</code> <code>ldap_rename_s()</code>	Chapter 7, "Adding, Modifying and Deleting Entries"

Most LDAP operations can be performed synchronously or asynchronously. The functions with names ending in `_s` are synchronous, and the remaining ones are asynchronous. For more information on the distinction between calling synchronous and asynchronous functions, see "Synchronous and Asynchronous Operations" on page 40 of Chapter 1, "Introduction" as well as "Synchronous and Asynchronous Functions" on page 75 of Chapter 4, "Using the LDAP API.")

Closing the Connection to the Server

When you have finished performing all necessary LDAP operations, you need to close the connection to the LDAP server. After you close the connection, you can no longer use the LDAP structure as it is freed from memory. To close a connection to an LDAP server, call one of the following functions:

- `ldap_unbind()`
- `ldap_unbind_s()`
- `ldap_unbind_ext()`

Both `ldap_unbind()` and `ldap_unbind_s()` are identical synchronous functions. They use different names so that each has a corresponding authentication function (`ldap_simple_bind()` and `ldap_simple_bind_s()`) to close the server connection. The `ldap_unbind_ext()` function allows you to explicitly include both server and client controls in your unbind request. However, since there is no server response to an unbind request, there is no way to receive a response from a server control that is included with your unbind request. Code Example 3-11 closes the current connection with the LDAP server.

Code Example 3-11 Closing an LDAP Server Connection

```
#include <stdio.h>
#include "ldap.h"
...
LDAP      *ld;
int        rc;
...
/* After completing your LDAP operations with the server, close
   the connection. */
rc = ldap_unbind( ld );
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_unbind: %s\n", ldap_err2string( rc ) );
}
...
```

Using the LDAP API

Sun™ Java System LDAP SDK for C includes a myriad of general functions commonly used when writing LDAP clients. This chapter includes information on some of these. It includes the following sections:

- Synchronous and Asynchronous Functions
- Retrieving SDK Information
- Managing Memory
- Reporting Errors
- Handling Referrals
- Setting Up an In-Memory Cache
- Handling Failover

Synchronous and Asynchronous Functions

You can perform most operations with *synchronous* or *asynchronous* functions. For example, to search the directory, you can call either the synchronous `ldap_search_ext_s()`, or the asynchronous `ldap_search_ext()`. In general, all synchronous functions have names ending with `_s`.

NOTE	The only difference between the synchronous and asynchronous functions are in the calling convention; the LDAP exchanges are identical.

Calling Synchronous Functions

When you call a *synchronous* function, your client waits for the operation to complete before executing any subsequent lines of code. Synchronous functions return LDAP_SUCCESS if successful and an LDAP error code if not. Code Example 4-1 calls the synchronous function `ldap_delete_ext_s()` to delete an entry in the directory.

Code Example 4-1 Calling Synchronous `ldap_delete_ext_s` to Delete an Entry

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

...
LDAP      *ld;
char      *matched_msg = NULL, *error_msg = NULL;
int       rc;
...
/* Perform an LDAP delete operation. */
rc = ldap_delete_ext_s( ld, DELETE_DN, NULL, NULL );
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_delete_ext_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
        fprintf( stderr, "%s\n", error_msg );
    }

    /* If the server cannot find an entry with the specified DN, it may send
    back the portion of the DN that matches an existing entry.*/
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
        fprintf( stderr,
            "Part of the DN that matches an existing entry: %s\n",
            matched_msg );
    }
} else {
    printf( "%s deleted successfully.\n", DELETE_DN );
}
...
```

To see other sample programs that call synchronous functions, open the source files in the `DSRK_base/lib/ldapsdk/examples` directory.

Calling Asynchronous Functions

When you call an *asynchronous* function, your client does not need to wait for the operation to complete. The client can continue performing other tasks (such as initiating other LDAP operations) while the asynchronous operation is executing. An asynchronous function passes back a unique message ID to identify the operation being performed. You can pass this message ID to the `ldap_result()` function to check the status of the operation. The following sections explain how to call an asynchronous function and check the results of the operation. To see other sample programs that call asynchronous functions, open the source files in the `DSRK_base/lib/ldapcsdk/examples` directory.

To Verify That an LDAP Request Was Sent

Asynchronous functions return an LDAP result code indicating whether or not the LDAP request was successfully sent to the server. If the function returns `LDAP_SUCCESS`, it has successfully sent the request to the server. Code Example 4-2 sends an LDAP delete request to the server and checks if the result was sent successfully. It uses the asynchronous `ldap_delete_ext()`.

Code Example 4-2 Calling Asynchronous `ldap_delete_ext` and Send Verification

```
#include <stdio.h>
#include "ldap.h"
...
/* Change these as needed. */
#define DELETE_DN "uid=wjensen,ou=People,dc=example,dc=com"
...
LDAP      *ld;
int        rc, msgid;
...
/* Send an LDAP delete request to the server. */
rc = ldap_delete_ext( ld, DELETE_DN, NULL, NULL, &msgid );
if ( rc != LDAP_SUCCESS ) {
    /* If the request was not sent successfully,
       print an error message and return. */
    fprintf( stderr, "ldap_delete_ext: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    return( 1 );
}
...
```

To Retrieve the Server Response

If the request was sent successfully, the function passes the message ID of the LDAP operation back to the client. You can use this message ID to determine if the server has sent back results for this operation by calling `ldap_result()`, and passing the message ID as a parameter. The function will then return one of the following values:

- -1 indicates that an error occurred.
- 0 indicates that the time-out period has been exceeded and the server has not yet sent a response back to your client.
- Any other value indicates that the server has sent a response for the requested operation back to your client. The `ldap_result()` parameter passes back a pointer to an `LDAPMessage` structure. This structure contains the server's response, which can include the following information:
 - An LDAP result code specifying the result of the operation you requested. See "Reporting Errors" on page 87 for details.
 - An additional error message sent back from the server. (This is optional.)
 - If the server was not able to find the entry specified by a DN, the portion of the DN that identifies an existing entry can be returned. See "Receiving the Matching Portion of a DN" on page 92 for details.
 - A set of referrals (if the server's directory does not contain the requested entries and if the server is configured to refer clients to other servers). See "Handling Referrals" on page 93 for details.
 - A set of server response controls applicable to the operation you requested. See Chapter 13, "Working with LDAP Controls" for information on controls.

NOTE	You can specify a time-out period to wait for results from the server.

Polling Loop

You can set up a loop to poll for results while doing other work. Code Example 4-3 defines a function that does work while waiting for the LDAP operation to complete and the server to send a response back to your client.

Code Example 4-3 Sample Loop to Poll for Results

```
int global_counter = 0;
void
do_other_work()
{
    global_counter++;
}
```

While Loop

Code Example 4-4 sets up a *while* loop to call your function when you are not checking for the server's response.

Code Example 4-4 While Loop to Call Function

```
#include <stdio.h>
#include "ldap.h"
...
LDAP          *ld;
LDAPMessage   *res;
LDAPControl   **serverctrls;
char          *matched_msg = NULL, *error_msg = NULL;
char          **referrals;
int           rc, parse_rc, msgid, finished = 0;
struct timeval zerotime;

zerotime.tv_sec = zerotime.tv_usec = 0L;
...
/* Send an LDAP delete request to the server. */
rc = ldap_delete_ext( ld, DELETE_DN, NULL, NULL, &msgid );
...
/* Poll the server for the results of the LDAP operation. */
while ( !finished ) {
    rc = ldap_result( ld, msgid, 0, &zerotime, &res );

    /* Check to see if a result was received. */
    switch ( rc ) {
        case -1:
            .../* An error occurred. */...
        case 0:
            /* The timeout period specified by zerotime was exceeded. This means that
               the server has still not yet sent the results of the delete
               operation back to your client. Break out of this switch statement,
               and continue calling ldap_result() to poll for results. */
            default:
                finished = 1;
                .../* Your client received a response from the server. */...
    }
}
```

Code Example 4-4 While Loop to Call Function (*Continued*)

```

/* Do other work while waiting. This is called if ldap_result() returns 0
(before you continue to top of the loop and call ldap_result() again). */
if ( !finished ) {
    do_other_work();
}
...
}
...

```

To Get Information from a Server Response

To get information from the server response, call `ldap_parse_result()` as illustrated in Code Example 4-5.

Code Example 4-5 Calling `ldap_parse_result()` Function

```

LDAP_API(int) LDAP_CALL
ldap_parse_result( LDAP *ld, LDAPMessage *res, int *errcodep,
char **matcheddn, char **errmsgp, char ***referralsp,
LDAPControl ***serverctrlsp, int freeit );

```

You can get the following information from parameters of `ldap_parse_result()`:

- `errcodep` holds the LDAP result code of the operation that the server finished processing. See “Reporting Errors” on page 87.
- `errmsgp` is an additional error message that the server can send to your client.
- `matcheddn` is the portion of the DN that matches an existing entry if the server is not able to find an entry for the DN that you’ve specified. See “Receiving the Matching Portion of a DN” on page 92.
- `referralsp` is a set of referrals sent back to your client if you’ve requested an entry that is not part of the directory information tree (DIT) managed by the server and if the server is configured to refer clients to other LDAP servers. See “Handling Referrals” on page 93.
- `serverctrlsp` is a set of server response controls applicable to the operation. See Chapter 13, “Working with LDAP Controls.”

NOTE When processing LDAP search operations, the server can send back individual entries matching the search, individual search references, and chains of entries and search references. For information on processing these types of results from the server, see Chapter 5, “Searching a LDAP Directory.”

Code Example 4-6 retrieves error information from an LDAPMessage structure returned by ldap_result().

Code Example 4-6 Retrieve Error Information from LDAPMessage Structure

```
#include <stdio.h>
#include "ldap.h"
...
LDAP          *ld;
LDAPMessage   *res;
LDAPControl   **serverctrls;
char          *matched_msg = NULL, *error_msg = NULL;
char          **referrals;
int           rc, parse_rc, msgid, finished = 0;
struct timeval zerotime;

zerotime.tv_sec = zerotime.tv_usec = 0L;
...
rc = ldap_result( ld, msgid, 0, &zerotime, &res );

/* Check to see if a result was received. */
switch ( rc ) {
case -1:
    ...
case 0:
    ...
default:
    ...
    /* Call ldap_parse_result() to get information from the results
       received from the server. */
    parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg,
                                &error_msg, &referrals, &serverctrls, 1 );

    /* Make sure the results were parsed successfully. */
    if ( parse_rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_parse_result: %s\n",
                 ldap_err2string( parse_rc ) );
        ldap_unbind( ld );
        return( 1 );
    }

    /* Check the results of the LDAP operation. */
    if ( rc != LDAP_SUCCESS ) {
        fprintf( stderr, "Error: %s\n", ldap_err2string(rc));
        if ( error_msg != NULL & *error_msg != '\0' ) {
            fprintf( stderr, "%s\n", error_msg );
        }
    }
}
```

Code Example 4-6 Retrieve Error Information from LDAPMessage Structure

```

/* If the server returned the portion of the DN
that identifies an existing entry, print it out. */
if ( matched_msg != NULL && *matched_msg != '\0' ) {
    fprintf( stderr,
        "Part of the DN that matches an existing entry: %s\n",
        matched_msg );
}
} else {
    printf( "Operation completed successfully" );
}
}
...

```

To Free LDAPMessage Structure

When you are done, you should call `ldap_msgfree()` to free the LDAPMessage structure (unless it is part of a chain of results). The result code returned by this function is not the same as the result code of the operation (`errcodep`). The result code returned by this operation indicates the type of operation from which the freed LDAPMessage structure is a response.

NOTE If you pass a non-zero value for the `freeit` parameter of `ldap_result()`, the structure is automatically freed after the information is retrieved.

To Cancel an Operation in Progress

If you need to cancel an LDAP operation, call `ldap_abandon_ext()`. The function returns `LDAP_SUCCESS` if successful or an LDAP result code if an error occurs. Once you cancel an operation, you cannot retrieve the results of that operation; thus, calling `ldap_result()` will not return any results.

Sample Code to Call an Asynchronous Function

Code Example 4-7 calls `ldap_delete_ext()` to delete an entry in the directory, and `ldap_result()` within a loop to poll the results of the delete.

Code Example 4-7 Asynchronous Function to Delete Entry Code

```

#include <stdio.h>
#include "ldap.h"
...
void do_other_work();
int global_counter = 0;

```

Code Example 4-7 Asynchronous Function to Delete Entry Code (*Continued*)

```

...
/* Change these as needed. */
#define DELETE_DN "uid=wjensen,ou=People,dc=example,dc=com"
...
LDAP          *ld;
LDAPMessage   *res;
LDAPControl   **serverctrls;
char          *matched_msg = NULL, *error_msg = NULL;
char          **referrals;
int           rc, parse_rc, msgid, finished = 0;
struct timeval zerotime;

zerotime.tv_sec = zerotime.tv_usec = 0L;
...
/* Send an LDAP delete request to the server. */
rc = ldap_delete_ext( ld, DELETE_DN, NULL, NULL, &msgid );
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_delete_ext: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    return( 1 );
}

/* Poll the server for the results of the delete operation. */
while ( !finished ) {
    /* Call ldap_result() to get the results of the delete operation.
       ldap_result() blocks for the time specified by the timeout argument
       (set to zerotime here) while waiting for the result from the server. */
    rc = ldap_result( ld, msgid, 0, &zerotime, &res );

    /* Check to see if a result was received. */
    switch ( rc ) {
    case -1:
        /* If ldap_result() returned -1, an error occurred. */
        rc = ldap_get_lderrno( ld, NULL, NULL );
        fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
        ldap_unbind( ld );
        return( 1 );

    case 0:
        /* The timeout period specified by zerotime was exceeded. This means that
           the server has still not yet sent the results of the delete operation
           back to your client. Break out of this switch statement, and
           continue calling ldap_result() to poll for results. */
        break;

    default:
        /* ldap_result() got the results of the delete operation
           from the server. No need to keep polling. */
        finished = 1;

        /* Call ldap_parse_result() to get information from the results
           received from the server. Note the last argument is a non-zero
           value. This means after the function retrieves information from the
           LDAPMessage structure, the structure is freed.
           (You don't need to call ldap_msgfree() to free the structure.)*/

```

Code Example 4-7 Asynchronous Function to Delete Entry Code (*Continued*)

```

    parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg,
        &error_msg, &referrals, &serverctrls, 1 );
    if ( parse_rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_parse_result: %s\n",
            ldap_err2string( parse_rc ) );
        ldap_unbind( ld );
        return( 1 );
    }

    /* Check the results of the LDAP delete operation. */
    if ( rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_delete_ext: %s\n", ldap_err2string(rc));
        if ( error_msg != NULL & *error_msg != '\0' ) {
            fprintf( stderr, "%s\n", error_msg );
        }
        /* Print the portion of a specified DN
           that matches an existing entry, if
           returned by the server. (For details, see
           "Receiving the Portion of the DN Matching an Entry.") */
        if ( matched_msg != NULL && *matched_msg != '\0' ) {
            fprintf( stderr,
                "Part of the DN that matches an existing entry: %s\n",
                matched_msg );
        }
    } else {
        printf( "%s deleted successfully.\n"
            "Counted to %d while waiting for the delete operation.\n",
            DELETE_DN, global_counter );
    }
}

/* Do other work while waiting for the results of the
   delete operation. */
if ( !finished ) {
    do_other_work();
}
}
ldap_unbind( ld );
return 0;
...

/* Perform other work while polling for results. */
void
do_other_work()
{
    global_counter++;
}
...

```

Retrieving SDK Information

You can get information about the particular version of the Sun Java System LDAP SDK for C that you are using by calling the `ldap_get_option()` function:

```
ldap_get_option (..., LDAP_OPT_API_INFO, ...);
```

The retrieved information can include the version of the SDK or the highest version of the LDAP that it supports. Code Example 4-8 illustrates how to use this function to retrieve version information.

Code Example 4-8 Retrieving SDK Version Information with `ldap_get_option()`

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

main()
{
    LDAPAPIInfo      ldapi;
    LDAPAPIFeatureInfo fi;
    int              i;
    int              rc;
    LDAP             *ld;

    memset( &ldapi, 0, sizeof(ldapi));
    ldapi.ldapai_info_version = LDAP_API_INFO_VERSION;

    if ((rc = ldap_get_option( ld, LDAP_OPT_API_INFO, &ldapi)) != 0) {
        printf("Error: ldap_get_option (rc: %d)\n", rc);
        exit(0);
    }

    printf("LDAP Library Information -\n"
        "    Highest supported protocol version: %d\n"
        "    LDAP API revision: %d\n"
        "    API vendor name: %s\n"
        "    Vendor-specific version: %.2f\n",
        ldapi.ldapai_protocol_version, ldapi.ldapai_api_version,
        ldapi.ldapai_vendor_name,
        (float)ldapi.ldapai_vendor_version / 100.0 );

    if ( ldapi.ldapai_extensions != NULL ) {
        printf("    LDAP API Extensions:\n");

        for ( i = 0; ldapi.ldapai_extensions[i] != NULL; i++ ) {
            printf("        %s", ldapi.ldapai_extensions[i] );
            fi.ldapaif_info_version = LDAP_FEATURE_INFO_VERSION;
            fi.ldapaif_name = ldapi.ldapai_extensions[i];
            fi.ldapaif_version = 0;

            if ( ldap_get_option( NULL, LDAP_OPT_API_FEATURE_INFO, &fi )
                != 0 ) {
                printf("Error: ldap_get_option( NULL,"
```

Code Example 4-8 Retrieving SDK Version Information with ldap_get_option()

```

        " LDAP_OPT_API_FEATURE_INFO, ... ) for %s failed"
        " (Feature Info version: %d)\n",
        fi.ldapaif_name, fi.ldapaif_info_version );
    } else {
        printf(" (revision %d)\n", fi.ldapaif_version);
    }
}
}
printf("\n");
}
```

Managing Memory

Several of the SDK functions allocate memory when called. When you have finished working with data allocated by these functions, you should free the memory. Table 4-1 lists some of the functions that allocate memory and the corresponding functions you use to free the memory when done.

NOTE For more information, see the descriptions of individual functions in Chapter 17, "Function Reference."

Table 4-1 SDK Functions to Allocate and Free Memory

Function to free memory	Type of memory freed
ldap_unbind() ldap_unbind_s()	Frees LDAP structures allocated by ldap_init().
ldap_msgfree()	Frees LDAPMessage structures allocated by ldap_result() or ldap_search_ext_s().
ldap_ber_free()	Frees BerElement structures allocated by ldap_first_attribute().
ldap_value_free()	Frees char ** arrays structures allocated by ldap_get_values().
ldap_value_free_len()	Frees arrays of berval structures allocated by ldap_get_values_len().
ber_bvfree()	Frees berval structures allocated by ldap_extended_operation_s(), ldap_parse_extended_result(), ldap_parse_sasl_bind_result(), and ldap_sasl_bind_s().
ldap_free_friendlymap()	Frees FriendlyMap structures allocated by ldap_friendly_name().

Table 4-1 SDK Functions to Allocate and Free Memory (*Continued*)

Function to free memory	Type of memory freed
<code>ldap_free_urldesc()</code>	Frees LDAPURLDesc structures allocated by <code>ldap_url_parse()</code> .
<code>ldap_getfilter_free()</code>	Frees LDAPFilterDesc structures allocated by <code>ldap_init_getfilter()</code> or <code>ldap_init_getfilter_buf()</code> .
<code>ldap_mods_free()</code>	Frees LDAPMod ** arrays and structures allocated by functions that you call when you add or modify entries.
<code>ldap_free_sort_keylist()</code>	Frees LDAPsortkey ** arrays structures allocated by calling <code>ldap_create_sort_keylist()</code> .
<code>ldap_control_free()</code>	Frees LDAPControl structures allocated by calling <code>ldap_create_sort_control()</code> or <code>ldap_create_persistentsearch_control()</code> .
<code>lldap_controls_free()</code>	Frees LDAPControl ** arrays structures allocated by calling <code>ldap_get_entry_controls()</code> , <code>ldap_parse_result()</code> , or <code>ldap_parse_reference()</code> .
<code>ldap_memfree()</code>	Any other types of memory that you allocate. (This function is a general function for freeing memory).

Reporting Errors

In the LDAP, the success or failure of an operation is specified by a *result code* sent back to the client. A result code of 0 normally indicates that the operation was successful whereas a non-zero result code usually indicates that an error occurred. For a detailed listing of result codes, see Chapter 18, “Result Codes.”

Setting Error Codes

When an LDAP operation is performed, the error information from the operation is specified in the LDAP structure. If you want to set error codes and error information in the LDAP structure, call the `ldap_set_lderrno()` function. Code Example 4-9 sets the LDAP_PARAM_ERROR error code in an LDAP structure.

Code Example 4-9 Setting an Error Code

```
#include "ldap.h"
...
LDAP *ld;
char *errmsg = "Invalid parameter";
...
if ( ldap_my_function() != LDAP_SUCCESS ) {
```

Code Example 4-9 Setting an Error Code

```

    ldap_set_lderrno( ld, LDAP_PARAM_ERROR, NULL, errmsg );
    return( 1 );
}
...

```

Getting Information About an Error

When an error occurs in an LDAP operation, the server sends an LDAP result code for the error back to the client as well as a message containing any additional information about the error. There are two ways you can get this information back from the server:

- If you are calling asynchronous functions, you can get the information from the LDAPMessage structure representing the result returned from the server. For details, see “To Get Information from an LDAPMessage Structure” on page 88.
- In situations where you do not have an LDAPMessage structure (for example, if you are calling functions that do not interact with the server), you can get error information from the LDAP structure (connection handle). For details, see “To Get Information from an LDAP Structure” on page 90.

To Get Information from an LDAPMessage Structure

If you have requested the operation through an asynchronous function, you can get the result of the operation from the server by calling the `ldap_result()` function. This function passes the result as an LDAPMessage structure. You can get information from this structure by calling the `ldap_parse_result()` function as in Code Example 4-10.

Code Example 4-10 Calling `ldap_parse_result()` Function

```

LDAP_API(int) LDAP_CALL ldap_parse_result( LDAP *ld,
LDAPMessage *res, int *errcodep, char **matcheddn,
char **errmsgp, char ***referralsp,
LDAPControl ***serverctrlsp, int freeit );

```

Different types of information are returned in the parameters of this function:

- The LDAP result code is the `errcodep` argument.

- Additional information from the server is passed back as the `errmsgp` argument.
- In cases when the server cannot find an entry from a DN, the portion of the DN that identifies an existing entry is passed back as the `matcheddn` argument. (See “Receiving the Matching Portion of a DN” on page 92 for details.)

NOTE You can also get the error message describing the LDAP result code by using the `ldap_err2string()` function. See “Getting the Error Message from an Error Code” on page 91 for details.

Code Example 4-11 will get and print information about an error returned from the server.

Code Example 4-11 Get and Print Error Information from LDAPMessage Structure

```
#include <stdio.h>
#include "ldap.h"
...
LDAP          *ld;
LDAPMessage   *res;
int           msgid = 0, rc = 0, parse_rc = 0, finished = 0;
char          *matched_msg = NULL, *error_msg = NULL;
char          **referrals;
LDAPControl   **serverctrls;
struct timeval zerotime;
...
while ( !finished ) {
    /* Check to see if the server returned a result. */
    rc = ldap_result( ld, msgid, 0, &zerotime, &res );
    switch ( rc ) {
        ...
        default:
            /* The client has received the result of the LDAP operation. */
            finished = 1;

            /* Parse this result to determine if the operation was successful.
            parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg,
            &error_msg, &referrals, &serverctrls, 1 );

            /* Verify that the result was parsed correctly. */
            if ( parse_rc != LDAP_SUCCESS ) {
                fprintf( stderr, "ldap_parse_result error: %s\n",
                    ldap_err2string( parse_rc ) );
                ldap_unbind( ld );
                return( 1 );
            }
        }

        /* Check the results of the operation. */
        if ( rc != LDAP_SUCCESS ) {
```

Code Example 4-11 Get and Print Error Information from LDAPMessage Structure (*Continued*)

```

/* Print the error message corresponding to the result code. */
fprintf( stderr, "Error: %s\n",
        ldap_err2string( rc ) );

/* If the server sent an additional message, print it out. */
if ( error_msg != NULL && *error_msg != '\0' ) {
    fprintf( stderr, "%s\n", error_msg );
}

/* If the server cannot find an entry with the specified DN,
   it may send back the portion of the DN that matches
   an existing entry. For details, see
   "Receiving the Portion of the DN Matching an Entry". */
if ( matched_msg != NULL && *matched_msg != '\0' ) {
    fprintf( stderr,
        "Part of the DN that matches an existing entry: %s\n",
        matched_msg );
}

/* Disconnect and return. */
ldap_unbind( ld );
return( 1 );
}
...

```

To Get Information from an LDAP Structure

In situations where you don't get an LDAPMessage structure (for example, if you are calling functions that do not interact with the server), you can get error information from the connection handle (the LDAP structure). You can get information about the last error that has occurred by calling the `ldap_get_lderrno()` function as in Code Example 4-12.

Code Example 4-12 Calling `ldap_get_lderrno()` Function

```

LDAP_API(int) LDAP_CALL ldap_get_lderrno(LDAP *ld,
char **m, char **s);

```

The different types of information retrieved are returned in the following ways:

- The LDAP result code is returned by this function.

- Additional information from the server is passed back as the *s* argument.
- In cases when the server cannot find an entry from a DN, the portion of the DN that identifies an existing entry is passed back as the *m* argument. (See “Receiving the Matching Portion of a DN” on page 92 for details.)

If you do not need to use the parameters returned by the `ldap_get_lderrno()` function, set the parameters to `NULL` as in:

```
ldap_get_lderrno( ld, NULL, NULL );
```

Code Example 4-13 gets and prints information about an error from an LDAP structure.

Code Example 4-13 To Get Error Message from an LDAP Structure

```
#include <stdio.h>
#include "ldap.h"
...
LDAP      *ld;
char*      *error_msg = NULL, *matched_msg = NULL;
int        rc;
...
rc = ldap_get_lderrno( ld, &matched_msg, &error_msg );
fprintf( stderr, "ldap_result error: %s\n", ldap_err2string( rc ) );
if ( error_msg != NULL && *error_msg != '\0' ) {
    fprintf( stderr, "%s\n", error_msg );
}

/* If the server cannot find an entry with the specified DN,
   it may send back the portion of the DN that matches
   an existing entry, For details, see
   "Receiving the Portion of the DN Matching an Entry". */
if ( matched_msg != NULL && *matched_msg != '\0' ) {
    fprintf( stderr,
        "Part of the DN that matches an existing entry: %s\n",
        matched_msg );
}
...
```

Getting the Error Message from an Error Code

If you have an error code and want to retrieve its corresponding error message, call the `ldap_err2string()` function which returns a pointer to the error message. The pointer returned by this function is a pointer to static data; do not free this string.

Code Example 4-14 Retrieving an Error Message from an Error Code

```
#include <stdio.h>
#include "ldap.h"
...
int      rc;
...
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "Error: %s\n", ldap_err2string( rc ) );
}
...
```

Receiving the Matching Portion of a DN

If an error occurred because an entry specified by a DN could not be found, the server may return the portion of the DN that identifies an existing entry. According to the LDAPv3, if a server returns an `LDAP_NO_SUCH_OBJECT`, `LDAP_ALIAS_PROBLEM`, `LDAP_INVALID_DN_SYNTAX`, or `LDAP_ALIAS_DEREF_PROBLEM` result code, the LDAP server should also send back the portion of DN that matches an entry that is closest to the requested entry. For example, suppose the LDAP server processes a request to modify the entry with the DN

`uid=bjensen,ou=Contractors,dc=example,dc=com` but that entry does not exist in the directory. If the DN `ou=Contractors,dc=example,dc=com` does exist, the server sends this portion of the DN with the result code `LDAP_NO_SUCH_OBJECT`. If the entry with the DN `ou=Contractors,dc=example,dc=com` doesn't exist either, but the entry with the DN `dc=example,dc=com` does, the server sends `dc=example,dc=com` back to the client with the result code `LDAP_NO_SUCH_OBJECT`. Basically, the server moves back up the directory tree (one DN component at a time) until it can find a DN that identifies an existing entry.

Printing Out Error Messages

To print out the error message describing the last error that occurred, call the `ldap_get_lderrno()` function. Code Example 4-15 prints a message if a function fails to delete an entry in the server.

Code Example 4-15 Printing Error Codes

```
#include "ldap.h"
...
int lderr;
```

Code Example 4-15 Printing Error Codes (*Continued*)

```

char * errmsg;
LDAP *ld;
char *dn = "uid=bjensen,ou=People,dc=example,dc=com";
...
if ( ldap_delete_s( ld, dn ) != LDAP_SUCCESS ) {
    lderr = ldap_get_lderrno (ld, NULL, &errmsg);
    if ( errmsg, != NULL ) {
        fprintf(stderr, "ldap_delete_s: %s\n", errmsg );
        ldap_memfree( errmsg );
    }
    return( 1 );
}
...

```

The client also prints out the following message if it does not have access permissions to delete the entry:

```
ldap_delete_s: Insufficient access
```

Handling Referrals

If an LDAP server receives a request for a DN that is not under its directory tree, it can refer clients to another LDAP server that may contain that DN; this is a *referral*. Suppose an LDAP server has a directory that starts under `dc=example,dc=com`. If your client sends the server a request to modify the entry with the DN `uid=bjensen,ou=People,dc=exampleWest,dc=com` (an entry that is not under `dc=example,dc=com`), one of the following may occur:

- If the server is not configured to send a referral, the server sends back an `LDAP_NO_SUCH_OBJECT` result code.
- If the server is configured to refer you to another LDAP server, the server sends a referral back to your client. This consists of the result code (`LDAP_PARTIAL_RESULTS` for LDAPv2 clients, or `LDAP_REFERRAL` for LDAPv3 clients) and one or more LDAP URLs. (For LDAPv2 clients, the URLs are included in the error message that the server sends to the client. For LDAPv3 clients, the URLs are included in a separate section of the result.)

Depending on how your LDAP client is configured, one of the following may occur:

- If your client handles referrals automatically, it connects to the LDAP server specified in the referral and requests the operation from that server. (The client binds anonymously to that server. To bind as a specific user, see “Binding For Referrals” on page 96.)
- If your client does not handle referrals automatically, it returns the result code sent from the server (LDAP_PARTIAL_RESULTS or LDAP_REFERRAL). You can get the LDAP URLs from the result by calling the `ldap_parse_result()` function.

NOTE	By default, clients built with Sun Java System LDAP SDK for C are configured to follow referrals automatically.

Search References and Referrals

A concept similar to a referral is a search reference. A *search reference* is an entry with the object class `referral`. The `ref` attribute of this object class contains an LDAP URL that identifies another LDAP server. When your client searches a subtree of a directory that contains search references, the server returns a mix of matching entries and search references. As your client retrieves search references from the server, one of the following occurs:

- If your client handles referrals automatically, the SDK retrieves each search reference, binds to the server identified in the reference, and retrieves the entry.
- If your client does not handle referrals automatically, the SDK simply adds the search reference to the chain of search results. The search reference is a message of the type `LDAP_RES_SEARCH_REFERENCE`. You can get the search references from a chain of results by calling the `ldap_first_reference()` and `ldap_next_reference()` functions. You can also call the `ldap_first_message()` and `ldap_next_message()` functions to get each message in the search results, and then call the `ldap_msgtype()` function to determine if the message is of the type `LDAP_RES_SEARCH_REFERENCE`.

NOTE	See “Binding For Referrals” on page 96 for information on specifying a DN and password for binding to a server for a referral.

Enabling or Disabling Referral Handling

By default, clients built with Sun Java System LDAP SDK for C automatically follow referrals to other servers. To change the way referrals are handled, call the `ldap_set_option()` function and pass `LDAP_OPT_REFERRALS` as the value of the option parameter.

- To prevent the client from automatically following referrals, set the `optdata` parameter to `LDAP_OPT_OFF`.
- If you want the client to automatically follow referrals again, set the `optdata` parameter to `LDAP_OPT_ON`.

Both `LDAP_OPT_OFF` and `LDAP_OPT_ON` are cast to `(void *)`. You can pass these parameters directly to the function as in Code Example 4-16. It prevents the client from automatically following referrals to other LDAP servers.

Code Example 4-16 Disabling Referrals

```
#include <stdio.h>
#include "ldap.h"
...
LDAP    *ld;
int      rc;
char     *host = "localhost";
...
/* Initialize a session with the LDAP server ldap.netscape.com:389. */
if ( ( ld = ldap_init( host, LDAP_PORT ) ) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
}

/* Never follow referrals. */
if ( ldap_set_option( ld, LDAP_OPT_REFERRALS, LDAP_OPT_OFF ) != LDAP_SUCCESS ) {
    rc = ldap_get_lderrno( ld, NULL, NULL );
    fprintf( stderr, "ldap_set_option: %s\n",
             ldap_err2string( rc ) );
    return( 1 );
}
...
```

Limiting Referral Hops

As a preference for the connection (or as a search constraint for specific search operations), you can specify the maximum number of referral hops that should be followed in a sequence of referrals. This is called the *referral hop limit*. For example, if your client is referred from LDAP server A to LDAP server B, from LDAP server

B to LDAP server C, and from LDAP server C to LDAP server D, your client is being referred 3 times in a row. If you set a limit of 2 referral hops, your client will not follow the referral to LDAP server D as this exceeds the referral hop limit. If the referral hop limit is exceeded, the result code `LDAP_REFERRAL_LIMIT_EXCEEDED` is returned.

To set the referral hop limit, pass `LDAP_OPT_REFERRAL_HOP_LIMIT` as the value of the option parameter and the maximum number of hops as the value of the optdata parameter. By default, the maximum number of hops is 5.

Binding For Referrals

If the session is set up so that referrals are always followed, the LDAP server that you connect to may refer you to another LDAP server. By default, the client binds anonymously (no user name or password specified) when following referrals. The following sections explain how to set up your client to authenticate with a DN and corresponding credentials.

How Referral Binding Works

If you want your client to authenticate to the referred LDAP server, you need to define a rebind function of the type `LDAP_REBINDPROC_CALLBACK` to get the DN and password to be used for authentication. Then, you specify that your function should be used if binding to other servers when following referrals. These steps detail the procedure.

1. The LDAP server sends a referral back to the client. The referral contains an LDAP URL that identifies another LDAP server.
2. The client calls the rebind function (specified by the `LDAP_OPT_REBIND_FN` option), passing 0 as the `freeit` argument.
3. The rebind function sets the `dn`, `passwd`, and `authmethodp` arguments.
 - o The `dn` argument points to the DN used to authenticate to the new LDAP server.
 - o The `passwd` argument points to the credentials for this DN.
 - o The `authmethodp` argument points to the method of authentication used (for example, `LDAP_AUTH_SIMPLE`).
4. If successful, the rebind function returns `LDAP_SUCCESS`, and the referral processing continues. If any other value is returned, referral processing stops, and that value is returned as the result code for the original LDAP request.

5. The client gets the DN, credentials, and authentication method from the arguments of the `rebind` function and uses this information to authenticate to the new LDAP server.
6. The client calls the `rebind` function again, passing `1` as the `freeit` argument.
7. The `rebind` function frees any memory allocated earlier to specify the DN and credentials.

Defining the Rebind Function

You need to define a `rebind` function that does the following:

- If `freeit` is `0`:
 - Set `dn` to point to the DN to be used for authentication.
 - Set `passwd` to point to the credentials to be used for authentication.
 - Set `authmethod` to point to the method of authentication to be used (for example, `LDAP_AUTH_SIMPLE`).

You can also make use of the `arg` argument, a pointer to the argument specified in the `ldap_set_rebind_proc()` function. If successful, it returns `LDAP_SUCCESS`; otherwise, it returns the appropriate LDAP error code.

- If `freeit` is `1`, free any memory that you allocated to create the DN and credentials.

Code Example 4-17 defines the `rebind` function and Table 4-2 its parameters. `LDAP_CALL` and `LDAP_CALLBACK` are used to set up calling conventions (for example, Pascal calling conventions on Windows) and are defined in the `lber.h` header file.

Code Example 4-17 Rebind Function Definition

```
int LDAP_CALL LDAP_CALLBACK rebindproc( LDAP *ld, char **dn,
    char **passwd, int *authmethod, int freeit, void *arg );
```

Table 4-2 LDAP_CALL and LDAP_CALLBACK parameters

Parameter Name	Description
<code>ld</code>	Pointer to the connection handle to the LDAP server.
<code>dn</code>	Pointer to the DN of the user (or entity) who wants to perform the LDAP operations. Your function needs to set this value.

Table 4-2 LDAP_CALL and LDAP_CALLBACK parameters (*Continued*)

Parameter Name	Description
passwdp	Pointer to the user's (or entity's) password. Your function needs to set this value.
authmethodp	Pointer to the method of authentication. Your function needs to set this value.
freeit	Specifies whether or not to free the memory allocated by the previous rebindproc() function call (in the event that this function is called more than once). If freeit is set to a non-zero value, your function should free the memory allocated by the previous call.
arg	Pointer to data that can be passed to your function.

Registering the Rebind Function

Once you have a rebind function, you need to register it. You can do this in one of the following ways:

- Call `ldap_set_rebind_proc()`, specifying your function and any data that you want passed as an argument.
- Call `ldap_set_option()` to set the `LDAP_OPT_REBIND_FN` option to your function and the `LDAP_OPT_REBIND_ARG` option to specify any arguments you want passed to your rebind function.

Setting Up an In-Memory Cache

The LDAP SDK for C includes functions that allow you to create an in-memory cache of search results for your client. Then, when sending a search request and receiving results, the results would be cached. The next time your client issues the same search request, the results are read from the cache. To set up a cache for your connection, complete the following steps illustrated in Code Example 4-18.

1. Call the `ldap_memcache_init()` function to create a new `LDAPMemCache` structure. This is the cache. Pass the pointer to this structure for subsequent operations.
2. Call the `ldap_memcache_set()` function to associate the cache with a connection handle (an LDAP structure).

Code Example 4-18 Creating an In-Memory Cache

```

#include "ldap.h"
...
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
...
LDAP          *ld;
LDAPMemCache  *dircache;
char *matched_msg = NULL, *error_msg = NULL;
int rc;
...
/* Get a handle to an LDAP connection. */
if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
}
...
/* Create an in-memory cache. */
rc = ldap_memcache_init( 0, 0, NULL, NULL, &dircache );
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_memcache_init: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    return( 1 );
}

/* Associate the cache with the connection. */
rc = ldap_memcache_set( ld, dircache );
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_memcache_set: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    return( 1 );
}
...

```

When a search request is cached, the search criteria is used as the key to the item in the cache. If you run the same search again, the results are read from the cache. If you alter the criteria (for example, specifying that you want all attributes returned instead of just the `uid` attribute), your client gets the results from the server rather than the cache.

The cache periodically checks for expired items and removes them from the cache. If you are writing a multi-threaded application and want to set up a separate thread to keep the cache up-to-date, you can call the `ldap_memcache_update()` function. If you want to remove items from the cache or flush it, call the `ldap_memcache_flush()` function. When you are done working with the cache, call the `ldap_memcache_destroy()` function.

Handling Failover

While performing an LDAP operation, if the LDAP client loses the connection with the server, the SDK returns an `LDAP_SERVER_DOWN` or `LDAP_CONNECT_ERROR` result code. To reconnect to the server, you can:

- Free the current connection handle and create a new connection handle. See “Creating a New Connection Handle” on page 100 for details.
- Use the reconnect option (`LDAP_OPT_RECONNECT`) to connect to the server again with the same connection handle. See “Using the Reconnect Option” on page 101 for details.

TIP You can use this option if you do not want to free the connection handle (for example, if multiple threads are sharing the same connection handle).

Creating a New Connection Handle

Call the `ldap_unbind()` or `ldap_unbind_s()` function to free the existing connection handle (the LDAP structure). Then call `ldap_init()` to initialize a new connection as in Code Example 4-19.

CAUTION The disadvantage to this approach is that you need to free the connection handle, which can make it difficult to share connection handles between threads.

Code Example 4-19 Initializing a New Connection Handle

```
#include "ldap.h"
...
LDAP *ld;
int tries = 0, rc = 0;
...
do {
    /* Call a function that performs an LDAP operation
    (my_ldap_request_function() can be any of these functions,
    such as ldap_search_ext_s()) */
    rc = my_ldap_request_function( ld );

    /* Check to see if the connection was lost. */
    if ( rc != LDAP_SERVER_DOWN && rc != LDAP_CONNECT_ERROR ) {
        return( rc ); /* Return result code. */
    }

    /* If the connection was lost, free the handle. */
```

Code Example 4-19 Initializing a New Connection Handle

```

ldap_unbind( ld );

/* Create a new connection handle and attempt to bind again. */
if ( ( ld = ldap_init( hostlist, port ) ) != NULL ) {
    ldap_simple_bind_s();

    /* Perform any other initialization
       work on the connection handle. */
}
} while ( ld != NULL && ++tries < 2 );
...

```

Using the Reconnect Option

To reconnect to the server without freeing the connection handle (for example, if multiple threads need to share the same connection handle), call the `ldap_set_option()` function to set the `LDAP_OPT_RECONNECT` option to `LDAP_OPT_ON`. Do this immediately after calling `ldap_init()` as detailed in Code Example 4-20.

Code Example 4-20 Reconnecting to an Existing Connection Handle

```

#include "ldap.h"
...
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
...
LDAP *ld;
...
/* Get a handle to an LDAP connection. */
if ( ( ld = ldap_init( HOSTNAME, PORTNUMBER ) ) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
}

/* Set the reconnect option. */
if ( ldap_set_option( ld, LDAP_OPT_RECONNECT, LDAP_OPT_ON ) == 0 ) {
    /* success */
} else {
    /* failure */
}
...

```

After setting this option, if the connection to the server has been lost (the SDK returns an `LDAP_CONNECT_ERROR` or `LDAP_SERVER_DOWN` result code to your client), call the `ldap_simple_bind_s()` function to reestablish a connection to one of the hosts specified in the `ldap_init()` function call. If your client is able to reconnect with the server, `ldap_simple_bind_s()` issues a bind request to the server and returns the result. Code Example 4-21 attempts to reconnect to the server if the client is disconnected.

Code Example 4-21 Reconnecting to Server After Disconnect

```
#include "ldap.h"
...
LDAP *ld;
int tries = 0, rc = 0;
...
do {
    /* Call a function that performs an LDAP operation
    (my_ldap_request_function() can be any of these functions,
    such as ldap_search_ext_s()) */
    rc = my_ldap_request_function( ld );

    /* Check to see if the connection was lost. */
    if ( rc != LDAP_SERVER_DOWN && rc != LDAP_CONNECT_ERROR ) {
        return( rc ); /* Return the result code. */
    }

    /* If the connection was lost, call
    ldap_simple_bind_s() to reconnect. */
    if ( ldap_simple_bind_s( ld, dn, passwd ) != LDAP_SUCCESS ) {
        /* failure -- could not reconnect */
        /* remember that ld is bad */
        return( rc );
    }
} while ( ++tries < 2 );
```

If the client has been successfully authenticated to the server using the current LDAP structure and the connection to the server has not been lost, `ldap_simple_bind_s()` returns `LDAP_SUCCESS` without contacting the LDAP server. The function is designed to do this in cases where more than one thread shares the same connection handle and receives an `LDAP_CONNECT_ERROR` or `LDAP_SERVER_DOWN` error. If multiple threads call `ldap_simple_bind_s()`, the function is designed so that only one of the threads actually issues the bind operation. For the other threads, the function returns `LDAP_SUCCESS` without contacting the LDAP server.

The important side effect to note is that if the `LDAP_OPT_RECONNECT` option is set, `ldap_simple_bind_s()` may return `LDAP_SUCCESS` without contacting the LDAP server. (The function only returns this if a bind with the DN was successfully performed in the past.)

Searching a LDAP Directory

This chapter explains how to call Sun™ Java System LDAP SDK for C functions to search a directory, retrieve the results, and get attributes and values from each entry in the search results. The chapter also provides examples of calling synchronous and asynchronous functions to search the directory. It contains the following sections:

- Overview
- Sending a Search Request
- Getting Search Results
- Sorting the Search Results
- Freeing the Search Results
- Search Examples

Overview

The LDAP SDK for C provides functions that allow you to search a directory and retrieve results from the server. For example, you can send a search request by calling the synchronous `ldap_search_ext_s()` or the asynchronous `ldap_search_ext()` and the server will send back matching results. In the LDAPv3, a server can send three different types of results, represented by `LDAPMessage` structures, back to the client:

- Directory entries found by the search.
- Search references found within the scope of the search. (A *search reference* is a reference to another LDAP server.)
- An LDAP result code specifying the result of the search operation.

NOTE In order to receive search references from LDAPv3 servers (such as Sun Java System Directory Server), you must identify your client as LDAPv3-enabled. If you do not, the server will return the LDAP error code `LDAP_PARTIAL_RESULTS` and a set of referrals. See “Specifying the LDAP Version of Your Client” on page 65 in Chapter 3, “Writing an LDAP Client” for details.

These results can be processed based on the following guidelines:

- If you are retrieving the results one at a time, you can call `ldap_result()` to get each result (an `LDAPMessage` structure) and determine the type of (entry or search reference).
- If you are retrieving a chain of results, you can call `ldap_first_message()` and `ldap_next_message()` to iterate through the results in the chain. If you are just interested in entries, you can call `ldap_first_entry()` and `ldap_next_entry()`. If you are just interested in search references, you can call `ldap_first_reference()` and `ldap_next_reference()`.
- To get an entry from a result (an `LDAPMessage` structure), call `ldap_first_entry()`.
- To get a search reference from a result (an `LDAPMessage` structure), call `ldap_parse_reference()`.
- To get the LDAP result code for the search operation from a result (an `LDAPMessage` structure), call `ldap_parse_result()`.

TIP To access data from entries found by the search, you need to follow this general process in your code:

1. Get each entry in the results.
2. Get the attributes from each entry.
3. Get the values from each attribute.

Sending a Search Request

To search the directory, call `ldap_search_ext_s()` or `ldap_search_ext()`. `ldap_search_ext_s()` is a synchronous function that blocks other work until all results have been received from the server. It is declared as illustrated in Code Example 5-1.

Code Example 5-1 ldap_search_ext_s() Search Request

```
LDAP_API(int) LDAP_CALL ldap_search_ext_s( LDAP *ld, const char *base,
int scope, const char *filter, char **attrs, int attrsonly,
LDAPControl **serverctrls, LDAPControl **clientctrls,
struct timeval *timeoutp, int sizelimit, LDAPMessage **res );
```

ldap_search_ext() is an asynchronous function that sends an LDAP search request to the server. You can do other work while checking to see if the server has returned any results. It is declared as illustrated in Code Example 5-2.

Code Example 5-2 ldap_search_ext() Search Request

```
LDAP_API(int) LDAP_CALL ldap_search_ext( LDAP *ld, const char *base,
int scope, const char *filter, char **attrs, int attrsonly,
LDAPControl **serverctrls, LDAPControl **clientctrls,
struct timeval *timeoutp, int sizelimit, int *msgidp );
```

Sample code for sending a search request can be found in “Sending A Search Request” on page 137.

Search Parameters

For either of the functions illustrated in Code Example 5-1 and Code Example 5-2, you specify the search criteria using the parameters as detailed in Table 5-1.

Table 5-1 Search Criteria Parameters

Parameter Name	Description
base	Specifies the starting point in the directory, or the base DN, where the search begins. For example, to search entries under dc=example,dc=com, the base DN is dc=example,dc=com. See “Specifying the Base DN and Scope” on page 108.
scope	Specifies which entries to search. You can narrow the scope of the search to search only the base DN, entries at one level under the base DN, or entries at all levels under the base DN. See “Specifying the Base DN and Scope” on page 108.

Table 5-1 Search Criteria Parameters

Parameter Name	Description
filter	Specifies a search filter by defining what to search for. A search filter can be as simple as “find entries where the last name is Jensen” or as complex as “find entries that belong to Dept. #17 and whose first names start with the letter F.” See “Defining a Search Filter” on page 109.
attrs attrsonly	Specify the type of information that you want returned (which attributes you want to retrieve) and whether you want the attribute type only or the attribute type and its values. You can also specify to return the names of attributes only (and not the values) by passing a non-zero value for the attrsonly argument. See “Specifying Attributes to Retrieve” on page 111.
serverctrls clientctrls	Specify the LDAPv3 controls associated with this search operation. For details on LDAPv3 controls, see Chapter 13, “Working with LDAP Controls.”
timeoutp sizelimit	Specify search constraints that you want applied to the search. For example, you can specify a different time-out period or maximum number of results from the values already defined for the current session. See “Setting Search Preferences” on page 112.

Specifying the Base DN and Scope

When sending a search request, you need to specify the base DN and scope of the search in order to identify the entries that you want searched. The base DN (the root argument) is the DN of the entry that serves as the starting point for the search. To specify the scope of the search, pass one of the following values as the scope parameter:

- LDAP_SCOPE_SUBTREE searches the base entry and all entries at all levels below the base entry (as illustrated in Figure 5-1).

Figure 5-1 Search Scope of LDAP_SCOPE_SUBTREE



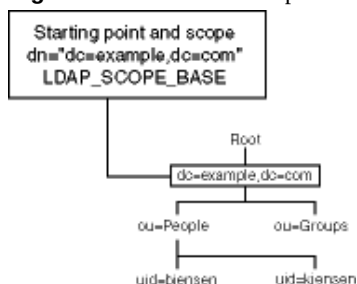
- `LDAP_SCOPE_ONELEVEL` searches all entries at one level below the base entry (as illustrated in Figure 5-2). The base entry is not included in the search. Use this setting if you just want a list of the entries under a given entry.

Figure 5-2 Search Scope of `LDAP_SCOPE_ONELEVEL`



- `LDAP_SCOPE_BASE` searches only the base entry. Use this setting if you just want to read the attributes of the base entry (as illustrated in Figure 5-3).

Figure 5-3 Search Scope of `LDAP_SCOPE_BASE`



Defining a Search Filter

When you search the directory, you use a search filter to define the search. Table 5-2 illustrates the search filter syntax and a simple example.

Table 5-2 Search Filter Syntax

Basic Search Filter Syntax	Example of Search Filter
<i>(attribute operator value)</i>	<i>(cn=Barbara Jensen)</i>

By comparing the syntax to the example we note that `cn` is the attribute, `=` is the operator, and `Barbara Jensen` is the value. The filter finds entries with the common name *Barbara Jensen*. For a listing of valid attributes that you can use in your search filter, see the documentation or LDAP schema for the directory server you are using. For a listing of valid operators that you can use in your search filter, see Table 5-3.

Table 5-3 Basic Operators for Search Filters

Operator	Description	Example
=	Returns entries whose attribute is equal to the value.	(cn=Barbara Jensen) finds the entry "cn=Barbara Jensen"
>=	Returns entries whose attribute is greater than or equal to the value.	(sn >= jensen) finds all entries from "sn=jensen" to "sn=z..."
<=	Returns entries whose attribute is less than or equal to the value.	(sn <= jensen) finds all entries from "sn=a..." to "sn=jensen"
=*	Returns entries that have a value set for that attribute.	(sn =*) finds all entries that have the sn attribute
~=	Returns entries whose attribute value approximately matches the specified value. Typically, this is an algorithm that matches words that sound alike.	(sn ~= jensen) finds the entry "sn = jansen"

Using boolean operators and parentheses, you can combine different sets of conditions into one filter. Table 5-4 illustrates the boolean search filter syntax for combining filters and a simple example.

Table 5-4 Boolean Search Filter Syntax

Boolean Search Filter Syntax	Example of Search Filter
(boolean_operator (filter1)(filter2)(filter3))	((sn=Jensen)(sn=Johnson))

The example uses the pipe (`|`) to signify a search for all entries with the last name *Jensen* or the last name *Johnson*. Table 5-5 is a listing of valid boolean operators that you can use in your search filter.

Table 5-5 Boolean Operators for Search Filters

Operator	Description
&	Returns entries matching all specified filter criteria.
	Returns entries matching one or more of the filter criteria.
!	Returns entries for which the filter is not true. You can only apply this operator to a single filter. For example, you can use: <code>(!(filter))</code> but not: <code>(!(filter1)(filter2))</code>

You can also include wildcards (*) to search for entries that start with, contain, or end with a given value. For example, you can use this filter to search for all entries whose names begin with the letter *F*:

```
(givenName=F*)
```

NOTE When comparing values containing letters, the value of the letter *a* is less than the value *z*. For example, the following filter finds all entries with last names beginning with *a* through *Jensen*:

```
(sn<=jensen)
```

Specifying Attributes to Retrieve

Using the `attrs` argument, you can either retrieve all attributes in the entries returned by the search, or you can specify the attributes from the search results that you want returned. To specify attributes, use the following guidelines:

- To return selected attributes, pass an array of the attribute names as the `attrs` argument. For example, to return only email addresses and phone numbers pass the NULL-terminated array `{"mail", "telephoneNumber", NULL}` as the `attrs` argument.
- To return all attributes in an entry, pass `NULL` as the `attrs` argument.
- To return none of the attributes from an entry, pass `LDAP_NO_ATTRS` as the `attrs` argument.

Sorting Attributes

If you plan to sort the results on your client, you need to return the attributes that you plan to use for sorting. For example, if you plan to sort by email address, make sure that the `mail` attribute is returned in the search results. See “Sorting the Search Results” on page 131 for more details.

Operational Attributes

Some attributes are used by servers for administering the directory. For example, the `creatorsName` attribute specifies the DN of the user who added the entry. These attributes are called *operational attributes*. Servers do not normally return operational attributes in search results unless you specify the attributes by name. For example, if you pass `NULL` for the `attrs` argument to retrieve all of the attributes in entries found by the search, the operational attribute `creatorsName` will not be returned to your client. You need to explicitly specify the `creatorsName` attribute in the `attrs` argument. To return all attributes in an entry and selected operational attributes, pass a `NULL`-terminated array containing `LDAP_ALL_USER_ATTRS` and the names of the operational attributes as the `attrs` argument. Table 5-6 lists some of the operational attributes and the information they contain.

Table 5-6 Information Available in Operational Attributes

Attribute Name	Description of Values
<code>createTimestamp</code>	The time the entry was added to the directory
<code>modifyTimestamp</code>	The time the entry was last modified
<code>creatorsName</code>	DN of the user who added the entry to the directory
<code>modifiersName</code>	DN of the user who last modified the entry
<code>subschemaSubentry</code>	DN of the subschema entry, which controls the schema for this entry. (See “Getting Schema Information” on page 217 of Chapter 10, “Retrieving Server Information” for details.)

Setting Search Preferences

For a given search, you can specify the maximum number of results to be returned or the maximum amount of time to wait for a search. Use the `timeoutp` and `sizelimit` arguments of the `ldap_search_ext_s()` or `ldap_search_ext()` functions with the following guidelines:

- To specify an infinite time limit (basically, no limit), create a `timeval` structure with `tv_sec = tv_usec = 0`, and pass a pointer to this as the `timeoutp` argument.
- To use the time limit specified by the `LDAP_OPT_TIMELIMIT` preference for this connection, pass `NULL` as the `timeoutp` argument.
- To specify an infinite size limit (basically, no limit), pass `LDAP_NO_LIMIT` as the `sizelimit` argument.
- To specify these preferences for all searches under the current connection, call `ldap_set_option()` and set the `LDAP_OPT_SIZELIMIT` and `LDAP_OPT_TIMELIMIT` options. If you do not want to specify a limit (basically, no limit), set the value of the option to `LDAP_NO_LIMIT`.

NOTE

The LDAP server may already have time and size constraints set up that you cannot override.

Code Example 5-3 sets these session preferences so that a search returns no more than 100 entries and takes no more than 30 seconds.

Code Example 5-3 Sample Code to Set Session Search Preferences

```
#include <stdio.h>
#include "ldap.h"
...
LDAP *ld;
int max_ret, max_tim;
char *host = "localhost";
...
/* Initialize a session with the LDAP server ldap.netscape.com:389. */
if ( ( ld = ldap_init( host, LDAP_PORT ) ) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
}

/* Set the maximum number of entries returned. */
max_ret = 100;
ldap_set_option(ld, LDAP_OPT_SIZELIMIT, (void *)&max_ret );

/* Set the maximum number of seconds to wait. */
max_tim = 30;
ldap_set_option( ld, LDAP_OPT_TIMELIMIT, (void *)&max_tim );
...
```

Getting Search Results

In the LDAPv3, the server returns search results as a chain of LDAPMessage structures with each structure containing:

- The directory entries found by the search. (Those that match the search criteria.)
- Any search references found within the scope of the search. (A *search reference* is a reference to another LDAP server).
- An LDAP result code specifying the result of the search operation.

NOTE	Because results are represented as a chain, you should not free individual LDAPMessage structures within the chain. When you are done working with the results, free the chain itself, rather than individual structures. Freeing individual LDAPMessage structures from memory may cause you to lose all of the results.

To retrieve an individual result from a chain of LDAPMessage structures, you can call one of the following sets of functions:

- To get each entry and search reference in the result, call `ldap_first_message()` and `ldap_next_message()`. Both of these functions return a pointer to an LDAPMessage structure that represents an entry, search reference, or LDAP result code. You can get the count of the structures in the chain by calling `ldap_count_messages()`.
- If you want to retrieve just the entries from the chain, call `ldap_first_entry()` and `ldap_next_entry()`. Both of these functions return a pointer to an LDAPMessage structure that represents an entry. You can get the count of the entries in the chain by calling `ldap_count_entries()`.
- If you want to retrieve just the search references from the chain, call `ldap_first_reference()` and `ldap_next_reference()`. Both of these functions return a pointer to an LDAPMessage structure that represents a search reference. You can get the count of the search references in the chain by calling `ldap_count_references()`.

Getting Results Synchronously

If you call `ldap_search_ext_s()` to search the directory synchronously, the function blocks processes until all results have been received. It then returns a chain of results in the `result` parameter (a handle to an `LDAPMessage` structure). Code Example 5-4 prints out the values of all attributes in the entries returned by a synchronous search.

Code Example 5-4 Synchronous Searching

```
#include <stdio.h>
#include "ldap.h"
/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
#define BASEDN "dc=example,dc=com"
#define SCOPE LDAP_SCOPE_SUBTREE
#define FILTER "(sn=Jensen)"
int
main( int argc, char **argv )
{
    LDAP          *ld;
    LDAPMessage    *res, *msg;
    LDAPControl    **serverctrls;
    BerElement     *ber;
    char           *a, *dn, *matched_msg = NULL, *error_msg = NULL;
    char           **vals, **referrals;
    int            version, i, rc, parse_rc, msgtype, num_entries = 0,
                  num_refs = 0;

    /* Get a handle to an LDAP connection. */
    if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
        perror( "ldap_init" );
        return( 1 );
    }
    version = LDAP_VERSION3;
    if ( ldap_set_option( ld, LDAP_OPT_PROTOCOL_VERSION, &version ) !=
        LDAP_SUCCESS ) {
        rc = ldap_get_lderrno( ld, NULL, NULL );
        fprintf( stderr, "ldap_set_option: %s\n", ldap_err2string( rc ) );
        ldap_unbind( ld );
        return( 1 );
    }
    /* Bind to the server anonymously. */
    rc = ldap_simple_bind_s( ld, NULL, NULL );
    if ( rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
        ldap_get_lderrno( ld, &matched_msg, &error_msg );
        if ( error_msg != NULL && *error_msg != '\0' ) {
            fprintf( stderr, "%s\n", error_msg );
        }
        if ( matched_msg != NULL && *matched_msg != '\0' ) {
            fprintf( stderr, "Part of the DN that matches an existing entry: %s\n",
                matched_msg );
        }
    }
}
```

Code Example 5-4 Synchronous Searching (*Continued*)

```

    }
    ldap_unbind_s( ld );
    return( 1 );
}
/* Perform the search operation. */
rc = ldap_search_ext_s( ld, BASEDN, SCOPE, FILTER, NULL, 0, NULL, NULL,
NULL, LDAP_NO_LIMIT, &res );
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_search_ext_s: %s\n", ldap_err2string( rc ) );
    if ( error_msg != NULL && *error_msg != '\0' ) {
        fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
        fprintf( stderr, "Part of the DN that matches an existing entry: %s\n",
matched_msg );
    }
    ldap_unbind_s( ld );
    return( 1 );
}
num_entries = ldap_count_entries( ld, res );
num_refs = ldap_count_references( ld, res );
/* Iterate through the results. An LDAPMessage structure sent back from
a search operation can contain either an entry found by the search,
a search reference, or the final result of the search operation. */
for ( msg = ldap_first_message( ld, res ); msg != NULL; msg =
ldap_next_message( ld, msg ) ) {
    /* Determine what type of message was sent from the server. */
    msgtype = ldap_msgtype( msg );
    switch( msgtype ) {
        /* If the result was an entry found by the search, get and print the
attributes and values of the entry. */
        case LDAP_RES_SEARCH_ENTRY:
            /* Get and print the DN of the entry. */
            if ( ( dn = ldap_get_dn( ld, res ) ) != NULL ) {
                printf( "dn: %s\n", dn );
                ldap_memfree( dn );
            }
            /* Iterate through each attribute in the entry. */
            for ( a = ldap_first_attribute( ld, res, &ber );
a != NULL; a = ldap_next_attribute( ld, res, ber ) ) {
                /* Get and print all values for each attribute. */
                if ( ( vals = ldap_get_values( ld, res, a ) ) != NULL ) {
                    for ( i = 0; vals[ i ] != NULL; i++ ) {
                        printf( "%s: %s\n", a, vals[ i ] );
                    }
                    ldap_value_free( vals );
                }
                ldap_memfree( a );
            }
            if ( ber != NULL ) {
                ber_free( ber, 0 );
            }
            printf( "\n" );
            break;
    }
}

```

Code Example 5-4 Synchronous Searching (*Continued*)

```

case LDAP_RES_SEARCH_REFERENCE:
    /* The server sent a search reference encountered during the
       search operation. */
    /* Parse the result and print the search references.
       Ideally, rather than print them out, you would follow the
       references. */
    parse_rc = ldap_parse_reference( ld, msg, &referrals, NULL, 0 );
    if ( parse_rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_parse_result: %s\n", ldap_err2string(
parse_rc ) );
        ldap_unbind( ld );
        return( 1 );
    }
    if ( referrals != NULL ) {
        for ( i = 0; referrals[ i ] != NULL; i++ ) {
            printf( "Search reference: %s\n\n", referrals[ i ] );
        }
        ldap_value_free( referrals );
    }
    break;
case LDAP_RES_SEARCH_RESULT:
    /* Parse the final result received from the server. Note the last
       argument is a non-zero value, which indicates that the
       LDAPMessage structure will be freed when done. (No need
       to call ldap_msgfree().) */
    parse_rc = ldap_parse_result( ld, msg, &rc, &matched_msg, &error_msg,
NULL, &serverctrls, 0 );
    if ( parse_rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_parse_result: %s\n", ldap_err2string(
parse_rc ) );
        ldap_unbind( ld );
        return( 1 );
    }
    /* Check the results of the LDAP search operation. */
    if ( rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_search_ext: %s\n", ldap_err2string( rc ) );
        if ( error_msg != NULL & *error_msg != '\0' ) {
            fprintf( stderr, "%s\n", error_msg );
        }
        if ( matched_msg != NULL && *matched_msg != '\0' ) {
            fprintf( stderr, "Part of the DN that matches an existing entry:
%s\n", matched_msg );
        }
    }
    else {
        printf( "Search completed successfully.\n"
            "Entries found: %d\n"
            "Search references returned: %d\n",
            num_entries, num_refs );
    }

    break;

default:
    break;

```

Code Example 5-4 Synchronous Searching (*Continued*)

```

    }
}
/* Disconnect when done. */
ldap_unbind( ld );
return( 0 );
}

```

Getting Results Asynchronously

If you use the asynchronous function `ldap_search_ext()`, you need to call `ldap_result()` (as illustrated in Code Example 5-5) to first determine if the server sent back any results.

Code Example 5-5 Declaring `ldap_result()`

```

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

```

You can specify how you want to get asynchronous results.

- To get the results one at a time (as the client receives them from the server), pass `LDAP_MSG_ONE` as the `all` argument. Code Example 5-6 is an example.

Code Example 5-6 Retrieving Search Results One at a Time

```

#include <stdio.h>
#include "ldap.h"
...
#define BASEDN "dc=example,dc=com"
#define SCOPE LDAP_SCOPE_SUBTREE
#define FILTER "(sn=Jensen)"
...
LDAP *ld;
LDAPMessage *res;
int msgid, rc, parse_rc, finished = 0;
struct timeval zertime;
zertime.tv_sec = zertime.tv_usec = 0L;
...
/* Send the LDAP search request. */
rc = ldap_search_ext( ld, BASEDN, SCOPE, FILTER, NULL, 0, NULL, NULL,
    NULL, LDAP_NO_LIMIT, &msgid );
...

```

Code Example 5-6 Retrieving Search Results One at a Time (*Continued*)

```

/* Poll the server for the results of the search operation. */
while ( !finished ) {
    rc = ldap_result( ld, msgid, LDAP_MSG_ONE, &zerotime, &res );
    switch ( rc ) {
        case -1:
            /* An error occurred. */
            ...
        case 0:
            /* The timeout period specified by zerotime was exceeded. */
            ...
        case LDAP_RES_SEARCH_ENTRY:
            /* The server sent one of the entries found by the search. */
            ...
        case LDAP_RES_SEARCH_REFERENCE:
            /* The server sent a search reference. */
            ...
        case LDAP_RES_SEARCH_RESULT:
            /* Parse the final result received from the server. */
            ...
    }
    ...
}
...

```

- To get the results all at once (in other words, to block processes until all results are received), pass `LDAP_MSG_ALL` as the `all` argument.
- To get the results received thus far, pass `LDAP_MSG_RECEIVED` as the `all` argument.

NOTE

If you specify either `LDAP_MSG_ALL` or `LDAP_MSG_RECEIVED`, the function passes back a chain of search results as the `result` argument. If you specify `LDAP_MSG_ONE`, the function passes back a single search result as the `result` argument. The function normally returns the type of the first search result; in this case, since only one result is returned, the function returns the type of that result.

Code Example 5-7 prints out the values of all attributes in the entries returned by an asynchronous search.

Code Example 5-7 Printing Results of Asynchronous Search

```

#include <stdio.h>
#include "ldap.h"
void do_other_work();
int global_counter = 0;

```

Code Example 5-7 Printing Results of Asynchronous Search (*Continued*)

```

/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
#define BASEDN "dc=example,dc=com"
#define SCOPE LDAP_SCOPE_SUBTREE
#define FILTER "(sn=Jensen)"
int
main( int argc, char **argv )
{
    LDAP          *ld;
    LDAPMessage    *res;
    BerElement     *ber;
    LDAPControl    **serverctrls;
    char          *a, *dn, *matched_msg = NULL, *error_msg = NULL;
    char          **vals, **referrals;
    int           version, i, msgid, rc, parse_rc, finished = 0,
                num_entries = 0, num_refs = 0;
    struct timeval zerotime;
    zerotime.tv_sec = zerotime.tv_usec = 0L;
    /* Get a handle to an LDAP connection. */
    if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
        perror( "ldap_init" );
        return( 1 );
    }
    version = LDAP_VERSION3;
    if ( ldap_set_option( ld, LDAP_OPT_PROTOCOL_VERSION, &version ) !=
        LDAP_SUCCESS ) {
        rc = ldap_get_lderrno( ld, NULL, NULL );
        fprintf( stderr, "ldap_set_option: %s\n", ldap_err2string( rc ) );
        ldap_unbind( ld );
        return( 1 );
    }
    /* Bind to the server anonymously. */
    rc = ldap_simple_bind_s( ld, NULL, NULL );
    if ( rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
        ldap_get_lderrno( ld, &matched_msg, &error_msg );
        if ( error_msg != NULL && *error_msg != '\0' ) {
            fprintf( stderr, "%s\n", error_msg );
        }
        /* If the server cannot find an entry,
           print the portion of the DN that matches
           an existing entry. (For details, see
           "Receiving the Portion of the DN Matching an Entry.") */
        if ( matched_msg != NULL && *matched_msg != '\0' ) {
            fprintf( stderr,
                "Part of the DN that matches an existing entry: %s\n",
                matched_msg );
        }
        ldap_unbind_s( ld );
        return( 1 );
    }
    /* Send the LDAP search request. */

```


Code Example 5-7 Printing Results of Asynchronous Search (*Continued*)

```

    rc = ldap_search_ext( ld, BASEDN, SCOPE, FILTER, NULL, 0, NULL, NULL,
NULL, LDAP_NO_LIMIT, &msgid );
    if ( rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_search_ext: %s\n", ldap_err2string( rc ) );
        ldap_unbind( ld );
        return( 1 );
    }
    /* Poll the server for the results of the search operation.
    Passing LDAP_MSG_ONE indicates that you want to receive
    the entries one at a time, as they come in. If the next
    entry that you retrieve is NULL, there are no more entries. */
    while ( !finished ) {
        rc = ldap_result( ld, msgid, LDAP_MSG_ONE, &zerotime, &res );
        /* The server can return three types of results back to the client,
        and the return value of ldap_result() indicates the result type:
        LDAP_RES_SEARCH_ENTRY identifies an entry found by the search,
        LDAP_RES_SEARCH_REFERENCE identifies a search reference returned
        by the server, and LDAP_RES_SEARCH_RESULT is the last result
        sent from the server to the client after the operation completes.
        You need to check for each of these types of results. */
        switch ( rc ) {
            case -1:
                /* An error occurred. */
                rc = ldap_get_lderrno( ld, NULL, NULL );
                fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
                ldap_unbind( ld );
                return( 1 );
            case 0:
                /* The timeout period specified by zerotime was exceeded.
                This means that the server has still not yet sent the
                results of the search operation back to your client.
                Break out of this switch statement, and continue calling
                ldap_result() to poll for results. */
                break;
            case LDAP_RES_SEARCH_ENTRY:
                /* The server sent one of the entries found by the search
                operation. Print the DN, attributes, and values of the entry. */
                /* Keep track of the number of entries found. */
                num_entries++;
                /* Get and print the DN of the entry. */
                if ( ( dn = ldap_get_dn( ld, res ) ) != NULL ) {
                    printf( "dn: %s\n", dn );
                    ldap_memfree( dn );
                }
                /* Iterate through each attribute in the entry. */
                for ( a = ldap_first_attribute( ld, res, &ber );
                    a != NULL; a = ldap_next_attribute( ld, res, ber ) ) {
                    /* Get and print all values for each attribute. */
                    if ( ( vals = ldap_get_values( ld, res, a ) ) != NULL ) {
                        for ( i = 0; vals[ i ] != NULL; i++ ) {
                            printf( "%s: %s\n", a, vals[ i ] );
                        }
                        ldap_value_free( vals );
                    }
                }
                ldap_memfree( a );
        }
    }

```

Code Example 5-7 Printing Results of Asynchronous Search (*Continued*)

```

    }
    if ( ber != NULL ) {
        ber_free( ber, 0 );
    }
    printf( "\n" );
    ldap_msgfree( res );
    break;
case LDAP_RES_SEARCH_REFERENCE:
    /* The server sent a search reference encountered during the
       search operation. */
    /* Keep track of the number of search references returned from
       the server. */
    num_refs++;
    /* Parse the result and print the search references.
       Ideally, rather than print them out, you would follow the
       references. */
    parse_rc = ldap_parse_reference( ld, res, &referrals, NULL, 1 );
    if ( parse_rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_parse_result: %s\n", ldap_err2string(
parse_rc ) );
        ldap_unbind( ld );
        return( 1 );
    }
    if ( referrals != NULL ) {
        for ( i = 0; referrals[ i ] != NULL; i++ ) {
            printf( "Search reference: %s\n\n", referrals[ i ] );
        }
        ldap_value_free( referrals );
    }
    break;
case LDAP_RES_SEARCH_RESULT:
    /* Parse the final result received from the server. Note the last
       argument is a non-zero value, which indicates that the
       LDAPMessage structure will be freed when done. (No need
       to call ldap_msgfree().) */
    finished = 1;
    parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg, &error_msg,
NULL, &serverctrls, 1 );
    if ( parse_rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_parse_result: %s\n", ldap_err2string(
parse_rc ) );
        ldap_unbind( ld );
        return( 1 );
    }
    /* Check the results of the LDAP search operation. */
    if ( rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_search_ext: %s\n", ldap_err2string( rc ) );
        ldap_get_lderrno( ld, &matched_msg, &error_msg );
        if ( error_msg != NULL & *error_msg != '\0' ) {
            fprintf( stderr, "%s\n", error_msg );
        }
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
        fprintf( stderr, "Part of the DN that matches an existing entry:
%s\n", matched_msg );
    }

```

Code Example 5-7 Printing Results of Asynchronous Search (*Continued*)

```

    }
    } else {
        printf( "Search completed successfully.\n"
               "Entries found: %d\n"
               "Search references returned: %d\n"
               "Counted to %d while waiting for the search operation.\n",
               num_entries, num_refs, global_counter );
    }

    break;

default:
    break;
}

/* Do other work here while waiting for the search operation to complete.
*/
if ( !finished ) {
    do_other_work();
}
/* Disconnect when done. */
ldap_unbind( ld );
return( 0 );
}
/*
 * Perform other work while polling for results. This doesn't do anything
 * useful, but it could.
 */
static void
do_other_work()
{
    global_counter++;
}

```

Determining Search Result Types

To determine what type of result was returned, call the `ldap_msgtype()` function. A search result can be one of the following types:

- `LDAP_RES_SEARCH_ENTRY` indicates that the result is an entry found in the search. You can pass the `LDAPMessage` structure representing the entry to `ldap_get_dn()` to get the DN of the entry or `ldap_first_attribute()` and `ldap_next_attribute()` to get the attributes of the entry. For details, see “Getting Distinguished Names for Each Entry” on page 125 and “Getting Attribute Names from an Entry” on page 127, respectively.

- `LDAP_RES_SEARCH_REFERENCE` indicates that the result is a search reference found within the scope of the search. You can pass the `LDAPMessage` structure representing the search reference to the `ldap_parse_reference()` function to get the referrals (LDAP URLs) to other servers. For details, see “Getting Referrals from Search References” on page 130.

NOTE

In order to receive search references from an LDAPv3 server (such as Sun Java System Directory Server), you must identify your client as LDAPv3-enabled. If not, the server will return the error code `LDAP_PARTIAL_RESULTS` and a set of referrals. See “Specifying the LDAP Version of Your Client” in Chapter 3, “Writing an LDAP Client” for details.

- `LDAP_RES_SEARCH_RESULT` indicates that the result is the final data sent by the server to indicate the finish of the LDAP search operation. You can pass the `LDAPMessage` structure representing the result to the `ldap_parse_result()` function to get the LDAP result code for the search operation. For a list of possible result codes for an LDAP search operation, see the documentation for the `ldap_search_ext_s()` function. For details on parsing the result, see “To Get Information from an LDAPMessage Structure” in Chapter 4, “Using the LDAP API.”

Code Example 5-8 retrieves each result in a chain and determines its type.

Code Example 5-8 Retrieving a Chain of Results

```
#include <stdio.h>
#include "ldap.h"
...
#define BASEDN "dc=example,dc=com"
#define SCOPE LDAP_SCOPE_SUBTREE
#define FILTER "(sn=Jensen)"
...
LDAP      *ld;
LDAPMessage *res, *msg;
BerElement *ber;
char      *matched_msg = NULL, *error_msg = NULL;
int        rc, msgtype, num_entries = 0, num_refs = 0;
...
/* Perform the search operation. */
rc = ldap_search_ext_s( ld, BASEDN, SCOPE, FILTER, NULL, 0, NULL, NULL,
    NULL, LDAP_NO_LIMIT, &res );
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_search_ext_s: %s\n", ldap_err2string( rc ) );
    if ( error_msg != NULL && *error_msg != '\0' ) {
        fprintf( stderr, "%s\n", error_msg );
    }
}
/* If the server cannot find an entry and returns the portion of the DN
that can find an entry, print it out. (For details, see
```

Code Example 5-8 Retrieving a Chain of Results (*Continued*)

```

    "Receiving the Portion of the DN Matching an Entry.") */
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
        fprintf( stderr,
            "Part of the DN that matches an existing entry: %s\n",
            matched_msg );
    }
    ldap_unbind_s( ld );
    return( 1 );
}
...
num_entries = ldap_count_entries( ld, res );
num_refs = ldap_count_references( ld, res );
...
/* Iterate through the results. */
for ( msg = ldap_first_message( ld, res ); msg != NULL;
      msg = ldap_next_message( ld, msg ) ) {

    /* Determine what type of message was sent from the server. */
    msgtype = ldap_msgtype( msg );
    switch( msgtype ) {
    case LDAP_RES_SEARCH_ENTRY:
        /* The result is an entry. */
        ...
    case LDAP_RES_SEARCH_REFERENCE:
        /* The result is a search reference. */
        ...
    case LDAP_RES_SEARCH_RESULT:
        /* The result is the final result sent by the server. */
        ...
    }
    ...
}
...

```

Getting Distinguished Names for Each Entry

Because the DN of an entry differentiates it from other entries, you may want to access the DN in search results. You may also want to parse the name into its individual components. The SDK provides functions for both of these tasks.

Getting the Distinguished Name of an Entry

To get the DN of an entry, call the `ldap_get_dn()` function. When finished with the DN, you should free it from memory by calling the `ldap_memfree()` function. Code Example 5-9 prints the DN for each entry found in a search.

Code Example 5-9 Obtaining DN for Search Result Entries

```

#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result, *e;
char *dn;
char *my_searchbase = "dc=example,dc=com";
char *my_filter = "(sn=Jensen)";
...
/* Search the directory. */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, my_filter,
    NULL, 0, &result ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_search_s" );
    return( 1 );
}

/* For each matching entry found, print the name of the entry.*/
for ( e = ldap_first_entry( ld, result ); e != NULL;
    e = ldap_next_entry( ld, e ) ) {
    if ( ( dn = ldap_get_dn( ld, e ) ) != NULL ) {
        printf( "dn: %s\n", dn );
        /* Free the memory used for the DN when done */
        ldap_memfree( dn );
    }
}
/* Free the result from memory when done. */
ldap_msgfree( result );

```

Getting the Components of a Distinguished Name

If you want to access individual components of a DN or relative DN, call the `ldap_explode_dn()` and `ldap_explode_rdn()` functions, respectively. Both functions return a NULL-terminated array containing the components of the DN. When you are done working with this array, free it by calling the `ldap_value_free()` function.

You can also specify whether or not you want the attribute names included in the array by using the `notypes` parameter. Set `notypes` to 0 if you want to include attribute names as in:

```
ldap_explode_dn( "uid=bjensen,ou=People,dc=example,dc=com", 0 )
```

This function then returns this array:

```
{ "uid=bjensen", "ou=People", "dc=example,dc=com", NULL }
```

Set `notypes` to 1 if you don't want to include the attribute names in the array as in:

```
ldap_explode_dn( "uid=bjensen,ou=People,dc=example,dc=com", 1 )
```

This function then returns this array:

```
{ "bjensen", "People", "example.com", NULL }
```

Getting Attribute Names from an Entry

To retrieve the name of the first attribute in an entry, call the `ldap_first_attribute()` function. To get the name of the next attribute, call the `ldap_next_attribute()` function.

NOTE Operational attributes such as `creatorsName` and `modifyTimestamp` are not normally returned in search results unless you explicitly specify them by name in the search request. For more details, see “Operational Attributes” on page 112.

When you are finished iterating through the attributes, you need to free the `BerElement` structure allocated by the `ldap_first_attribute()` function, if the structure is not `NULL`. To free this structure, call the `ldap_ber_free()` function. You should also free the attribute name returned by the `ldap_first_attribute()` function. To free the attribute name, call the `ldap_memfree()` function. Code Example 5-10 illustrates how to retrieve each attribute of an entry.

Code Example 5-10 Retrieving Entry Attributes

```
#include <stdio.h>
#include <ldap.h>
...
LDAP      *ld;
LDAPMessage *result, *e;
BerElement *ber;
char      *a;
char      *my_searchbase = "dc=example,dc=com";
char      *my_filter = "(sn=Jensen)";
...
/* Search the directory. */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, my_filter,
    NULL, 0, &result ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_search_s" );
    return( 1 );
}

/* Get the first matching entry.*/
e = ldap_first_entry( ld, result );

/* Retrieve the attributes of the entry. */
for ( a = ldap_first_attribute( ld, e, &ber ); a != NULL;
```

Code Example 5-10 Retrieving Entry Attributes (*Continued*)

```

        a = ldap_next_attribute(ld, e, ber)){
    ...
    /* Code to get and manipulate attribute values */
    ...
    }
    ldap_memfree( a );
}
/* Free the BerElement structure from memory when done. */
if ( ber != NULL ) {
    ldap_ber_free( ber, 0 );
}
...

```

Getting the Values of an Attribute

The values of an attribute are represented by a NULL-terminated array. The values are either a list of strings (if the attribute contains string data, such as a name or phone number) or a list of `berval` structures (if the attribute contains binary data, such as a JPEG file or an audio file). Use the following guidelines to retrieve the values of an attribute.

- To get the values of an attribute that contains string data, call the `ldap_get_values()` function. `ldap_get_values()` returns a NULL-terminated array of strings representing the value of the attribute.
- To get the values of an attribute that contains binary data, call the `ldap_get_values_len()` function. `ldap_get_values_len()` returns a NULL-terminated array of `berval` structures representing the value of the attribute.
- To get the number of values in an attribute, call either the `ldap_count_values()` or `ldap_count_values_len()` function. Both functions return the number of values in the attribute.

When you have finished working with the values of the attribute, you need to free them from memory. To do this, call `ldap_value_free()` or `ldap_value_free_len()`. Code Example 5-11 gets and prints the values of an attribute in an entry. It assumes that all attributes have string values.

Code Example 5-11 Retrieving Attribute Values

```

#include <stdio.h>
#include <ldap.h>
...

```


Code Example 5-11 Retrieving Attribute Values (*Continued*)

```

LDAP          *ld;
LDAPMessage   *result, *e;
BerElement    *ber;
char          *a;
char          **vals;
char          *my_searchbase = "dc=example,dc=com";
char          *my_filter = "(sn=Jensen)";
int i;
...
/* Search the directory. */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, my_filter,
    NULL, 0, &result ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_search_s" );
    return( 1 );
}

/* Get the first matching entry. */
e = ldap_first_entry( ld, result );

/* Get the first matching attribute. */
a = ldap_first_attribute( ld, e, &ber );

/* Get the values of the attribute. */
if ( ( vals = ldap_get_values( ld, e, a ) ) != NULL ) {
    for ( i = 0; vals[i] != NULL; i++ ) {
        /* Print the name of the attribute and each value */
        printf( "%s: %s\n", a, vals[i] );
    }
    /* Free the attribute values from memory when done. */
    ldap_value_free( vals );
}
...

```

Code Example 5-12 gets the first value of the jpegPhoto attribute and saves the JPEG data to a file.

Code Example 5-12 Getting and Saving Attribute Value

```

#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result, *e;
BerElement *ber;
char *a;
struct berval photo_data;
struct berval **list_of_photos;
FILE *out;
char *my_searchbase = "dc=example,dc=com";

```

Code Example 5-12 Getting and Saving Attribute Value (*Continued*)

```

char *my_filter = "(sn=Jensen)";
...
/* Search the directory. */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, my_filter,
    NULL, 0, &result ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_search_s" );
    return( 1 );
}

/* Get the first matching entry.*/
e = ldap_first_entry( ld, result );

/* Find the jpegPhoto attribute. */
a = ldap_first_attribute( ld, e, &ber );
while ( strcasecmp( a, "jpegphoto" ) != 0 ) {
    a = ldap_next_attribute( ld, e, ber );
}

/* Get the value of the attribute. */
if ( ( list_of_photos = ldap_get_values_len( ld, e, a ) ) != NULL ) {
    /* Prepare to write the JPEG data to a file */
    if ( ( out = fopen( "photo.jpg", "wb" ) ) == NULL ) {
        perror( "fopen" );
        return( 1 );
    }
    /* Get the first JPEG. */
    photo_data = *list_of_photos[0];
    /* Write the JPEG data to a file */
    fwrite( photo_data.bv_val, photo_data.bv_len, 1, out );
    fclose( out );
    /* Free the attribute values from memory when done. */
    ldap_value_free_len( list_of_photos );
}
...

```

Getting Referrals from Search References

A search reference returned from the server contains one or more referrals (LDAP URLs that identify other LDAP servers). To retrieve these referrals, you need to call the `ldap_parse_reference()` function. Code Example 5-13 gets and prints the referrals from a search reference.

Code Example 5-13 Obtaining a Referral

```

#include <stdio.h>
#include "ldap.h"
...
LDAP      *ld;

```

Code Example 5-13 Obtaining a Referral (*Continued*)

```

LDAPMessage *msg;
char **referrals;
int i, rc, parse_rc;
...
parse_rc = ldap_parse_reference( ld, msg, &referrals, NULL, 0 );
if ( parse_rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_parse_result: %s\n",
            ldap_err2string( parse_rc ) );
    ldap_unbind( ld );
    return( 1 );
}
if ( referrals != NULL ) {
    for ( i = 0; referrals[ i ] != NULL; i++ ) {
        printf( "Search reference: %s\n\n", referrals[ i ] );
    }
    ldap_value_free( referrals );
}
...

```

Sorting the Search Results

The LDAP SDK for C contains functions to sort entries and values in the search results. You can either specify that the server return sorted results or you can sort entries on your client.

Server-Side Sorting

To sort results on the server, you need to send a server-side sorting control with the search request. For details, see “Using the Server-Side Sorting Control” in Chapter 13, “Working with LDAP Controls.”

Client-Side Sorting

First, you need to retrieve the attributes that you plan to use for sorting. For example, if you plan to sort the results by email address, make sure that the `mail` attribute is one of the attributes returned in the search.

Sorting Entries by an Attribute

To sort the search results by a particular attribute, call the `ldap_sort_entries()` function. If you don't specify an attribute for sorting (that is, if you pass `NULL` for the `attr` parameter), the entries are sorted by DN. Code Example 5-14 sorts entries by the `roomNumber` attribute:

Code Example 5-14 Sorting Entries by Attribute Value

```
#include <stdio.h>
#include <string.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result;
char *my_searchbase = "dc=example,dc=com";
char *my_filter = "(sn=Jensen)";
char *sortby = "roomNumber";
...
/* Search the directory. */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, my_filter,
    NULL, 0, &result ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_search_s" );
    return( 1 );
}

/* Sort the results by room number, using strcasecmp. */
if (ldap_sort_entries(ld, &result, sortby, strcasecmp) != LDAP_SUCCESS){
    ldap_perror( ld, "ldap_sort_entries" );
    return( 1 );
}
...
```

Sorting Entries by Multiple Attributes

To sort the search results by multiple attributes, call the `ldap_multisort_entries()` function. If you don't specify a set of attributes for sorting (that is, if you pass `NULL` for the `attr` parameter), the entries are sorted by DN. Code Example 5-15 sorts entries first by the `roomNumber` attribute, then by the `telephoneNumber` attribute.

Code Example 5-15 Sorting Entries by Multiple Values

```
#include <stdio.h>
#include <string.h>
#include <ldap.h>
LDAP *ld;
LDAPMessage *res;
```

Code Example 5-15 Sorting Entries by Multiple Values

```

char *my_searchbase = "dc=example,dc=com";
char *my_filter = "(sn=Jensen)";
char *attrs[2];
attrs[0] = "roomNumber";
attrs[1] = "telephoneNumber";
attrs[2] = NULL;
...
/* Search the directory. */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, my_filter,
    NULL, 0, &res ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_search_s" );
    return( 1 );
}

/* Sort the results, using strcasecmp. */
if ( ldap_multisort_entries( ld, &res, attrs, strcasecmp ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_sort_entries" );
    return( 1 );
}

```

Sorting the Values of an Attribute

You can also sort the values of a particular attribute. To do this, call the `ldap_sort_strcasecmp()` function. In this function, the comparison function must pass parameters of the type `char **`. You should use the `ldap_sort_strcasecmp()` function, rather than a function like `strcasecmp()` (which passes parameters of the type `char *`). Code Example 5-16 sorts the values of attributes before printing them.

Code Example 5-16 Sorting Attribute Values

```

#include <stdio.h>
#include <string.h>
#include <ldap.h>
LDAP *ld;
LDAPMessage *result, *e;
BerElement *ber;
char *a, *dn;
char **vals;
int i;
char *my_searchbase = "dc=example,dc=com";
char *my_filter = "(sn=Jensen)";
...
if ( ( vals = ldap_get_values( ld, e, a ) ) != NULL ) {
    /* Sort the values of the attribute */
    if ( ldap_sort_values( ld, vals, strcasecmp ) != LDAP_SUCCESS ) {
        ldap_perror( ld, "ldap_sort_values" );
        return( 1 );
    }
}

```

Code Example 5-16 Sorting Attribute Values (*Continued*)

```

    }
    /* Print the values of the attribute. */
    for ( i = 0; vals[i] != NULL; i++ ) {
        printf( "%s: %s\n", a, vals[i] );
    }
    /* Free the values from memory. */
    ldap_value_free( vals );
}
...

```

Freeing the Search Results

The results of the search are returned in an LDAPMessage structure. After you are done working with the search results, you should free this structure from memory. To free the search results, call the `ldap_msgfree()` function which returns the type of the last message freed from memory.

Search Examples

This section contains sample code for some search operations.

Reading an Entry with a Search

You can use the search functions to read a specific entry in the directory. To read an entry, set the starting point of the search to the entry, and set the scope of the search to `LDAP_SCOPE_BASE`, specifying `(objectclass=*)` as the search filter. Code Example 5-17 prints the attributes of the entry for Barbara Jensen.

Code Example 5-17 Reading a Specific Entry with a Search

```

#include <stdio.h>
#include "ldap.h"
/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORT_NUMBER LDAP_PORT
#define FIND_DN "uid=bjensen,ou=People,dc=example,dc=com"
int
main( int argc, char **argv )
{
    LDAP *ld;

```

Code Example 5-17 Reading a Specific Entry with a Search (*Continued*)

```

LDAPMessage *result, *e;
BerElement *ber;
char *a;
char **vals;
int i, rc;
/* Get a handle to an LDAP connection. */
if ( (ld = ldap_init( HOSTNAME, PORT_NUMBER )) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
}
/* Bind anonymously to the LDAP server. */
if ( ( rc = ldap_simple_bind_s( ld, NULL, NULL ) ) != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
    return( 1 );
}
/* Search for the entry. */
if ( ( rc = ldap_search_ext_s( ld, FIND_DN, LDAP_SCOPE_BASE,
"(objectclass=*)",
NULL, 0, NULL, NULL, LDAP_NO_LIMIT, LDAP_NO_LIMIT, &result ) ) !=
LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_search_ext_s: %s\n", ldap_err2string( rc ) );
    return( 1 );
}
/* Since we are doing a base search, there should be only one matching
entry. */
e = ldap_first_entry( ld, result );
if ( e != NULL ) {
    printf( "\nFound %s:\n\n", FIND_DN );
    /* Iterate through each attribute in the entry. */
    for ( a = ldap_first_attribute( ld, e, &ber );
a != NULL; a = ldap_next_attribute( ld, e, ber ) ) {
        /* For each attribute, print the attribute name and values. */
        if ( (vals = ldap_get_values( ld, e, a )) != NULL ) {
            for ( i = 0; vals[i] != NULL; i++ ) {
                printf( "%s: %s\n", a, vals[i] );
            }
            ldap_value_free( vals );
        }
        ldap_memfree( a );
    }
    if ( ber != NULL ) {
        ber_free( ber, 0 );
    }
}
ldap_msgfree( result );
ldap_unbind( ld );
return( 0 );
}

```

Listing Subentries with a Search

You can use the search functions to list the subentries under a specific entry in the directory. To list the subentries, set the starting point of the search to the entry, and set the scope of the search to `LDAP_SCOPE_ONELEVEL`. Code Example 5-18 lists all entries one level below the `dc=example,dc=com` entry in the directory hierarchy.

Code Example 5-18 Listing Subentries

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

LDAP      *ld;
LDAPMessage *result, *e;
BerElement *ber;
char      *a, *dn;
char      **vals;
char      *my_searchbase = "dc=example,dc=com";
char      *my_filter = "(objectclass=*)"

/* Search one level under the starting point. */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_ONELEVEL, my_filter,
    NULL, 0, &result ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_search_s" );
    return( 1 );
}
/* For each matching entry, print the entry name and its attributes. */
for ( e = ldap_first_entry( ld, result ); e != NULL;
    e = ldap_next_entry( ld, e ) ) {
    if ( ( dn = ldap_get_dn( ld, e ) ) != NULL ) {
        printf( "dn: %s\n", dn );
        ldap_memfree( dn );
    }
    for ( a = ldap_first_attribute( ld, e, &ber ); a != NULL;
        a = ldap_next_attribute( ld, e, ber ) ) {
        if ( ( vals = ldap_get_values( ld, e, a ) ) != NULL ) {
            for ( i = 0; vals[i] != NULL; i++ ) {
                printf( "%s: %s\n", a, vals[i] );
            }
            ldap_value_free( vals );
        }
        ldap_memfree( a );
    }
    if ( ber != NULL ) {
        ldap_ber_free( ber, 0 );
    }
    printf( "\n" );
}
ldap_msgfree( result );
...
```


Sending A Search Request

Code Example 5-19 illustrates how to send a search request to the server for all entries with the last name (or surname) *Jensen* in the *example.com* organization.

Code Example 5-19 Sample Code to Send a Search Request

```
#include <stdio.h>
#include "ldap.h"
...
#define BASEDN "dc=example,dc=com"
#define SCOPE LDAP_SCOPE_SUBTREE
#define FILTER "(sn=Jensen)"
...
LDAP      *ld;
int        msgid, rc;
...
/* Send the search request. */
rc = ldap_search_ext( ld, BASEDN, SCOPE, FILTER, NULL, 0, NULL,
    NULL, NULL, LDAP_NO_LIMIT, &msgid );
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_search_ext: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    return( 1 );
}
...
```


Using Filter Configuration Files

In addition to the search functions detailed in Chapter 5, “Searching a LDAP Directory,” Sun™ Java System LDAP SDK for C contains functions for integrating filter configuration files into the search. This chapter explains about filter configuration files and how to use them with the SDK. It contains the following sections:

- Filter Configuration Files
- Loading a Filter Configuration File
- Filter Prefixes and Suffixes
- Freeing Filters from Memory
- Creating Filters Programmatically

Filter Configuration Files

Filter configuration files can help simplify the process of selecting the appropriate filter for a search request. A *filter configuration file* contains a list of filters that you can load and use in your searches. If you are writing a client that allows users to search the directory, you might want to use different search filters tailored for specific types of search criteria. For example, if the user wants to search for this email address:

`bjensen@example.com`

You might want to use this search filter:

`(mail=bjensen@example.com)`

Similarly, suppose the search term entered by the user contains numbers, as in:

`555-1212`

In this case, you might want to use this search filter:

(telephoneNumber=555-1212)

Rather than write code to find and select the appropriate filter, you can include the filters in a filter configuration file. For example, Code Example 6-1 is a section of a filter configuration file specifying one filter for telephone numbers and two filters for email addresses. The telephone number filter is used if the search criteria contains one or more numbers. The email filters are used if the search criteria contains an “at” sign (@).

Code Example 6-1 Section of Filter Configuration File

```
"people"
  "^[0-9][0-9-]*$"           " "
  "(telephoneNumber=%v))"    "phone number ends with"
  "@ " " " "(mail=%v)"      "email address is"
  "(mail=%v*)"              "email address starts with"
```

NOTE You should specify the filters in the order that you want them to be used. For example, if you want to apply the (mail=%v) filter before the (mail=%v*) filter, make sure that the filters appear in that order.

The Filter Configuration File Syntax

A filter configuration file has the format illustrated in Code Example 6-2. The variables are discussed in the sections following.

Code Example 6-2 Syntax for Filter Configuration File

```
tag
  pattern1    delimiters    filter1-1  desc1-1  [scope]
  filter1-2   desc1-2
  pattern2    delimiters    filter2-1  desc2-1  [scope]
  ...
```

Tag

A *tag* identifies a group of filters. You can use different tags to distinguish filters for different types of objects. For example, you can use one tag to represent filters for person entries, another tag to represent filters for organization entries, and so on.

```
"people"
    ... (filters for searching "person" entries) ...
"organization"
    ... (filters for "organization" entries) ...
```

When you call functions like `ldap_getfirstfilter()` to retrieve a filter, you can specify a tag (or part of a tag) as a parameter. The tag narrows the list of filters that the function can retrieve.

Patterns

pattern1 and *pattern2* are regular expressions used to determine which filter is selected, based on the search criteria. For example, if you specify `"^[0-9]"` as the pattern for a filter, the filter is selected for all search criteria beginning with a number.

```
"people"
    "^[0-9]"      ...
```

NOTE

For more information on regular expressions, consult your UNIX[®] documentation. (For example, documentation on the `ed` command contains some information on regular expressions.)

Delimiters

Delimiters specifies the delimiters used to distinguish one field from another within the search criteria. For example, if the search criteria consists of a city name and state abbreviation separated by a comma, specify “,” as the delimiter.

Filters

filter1-1, *filter1-2*, and *filter2-1* are filters. Use `%v` to represent the search criteria. For example, to search e-mail addresses, use the filter (`mail=%v`). During runtime, if the search criteria `bjensen@example.com` is entered, the filter becomes (`mail=bjensen@example.com`).

If the search criteria consists of a number of delimited fields (for example, a "last name, first name" format like "Jensen, Barbara"), use %v1, %v2, . . . , %vn to represent the different fields within the search criteria as in:

```
"people"
"^[A-Z]*, "      " "      (&(sn=%v1)(givenName=%v2))
```

In this example, the delimiter is a comma. The word before the delimiter replaces %v1 in the filter, and the word after the delimiter replaces %v2 in the filter. If the user searches for:

```
Jensen, Barbara
```

The resulting filter is:

```
(&(sn=Jensen)(givenName=Barbara))
```

You can also specify ranges of fields. For example, to specify the values in the first three fields, use %v1-3. To specify values from the third field to the last field, use %v3-. To specify the value in the last field, use %v\$.

Descriptions

desc1-1, *desc1-2*, and *desc2-1* are phrases briefly describing the filters.

Filter Parameters

Table 6-1 lists the parameters that can be used within a filter.

Table 6-1 Filter Parameters

Parameter	Description
%v	This parameter means that the entire search criteria is inserted in place of %v. For example, if the filter is (mail=%v), entering the word jensen results in the filter (mail=jensen).
%v\$	This parameter means that the last word in the search criteria is inserted in place of %v. For example, if the filter is (sn=%v), entering the words Barbara Jensen results in the filter (sn=Jensen).
%vN	N is a single digit between 1 and 9. This parameter means that the Nth word in the search criteria is inserted in place of %vN. For example, if the filter is (sn=%v2), entering the words Barbara Jensen results in the filter (sn=Jensen).
%vM-N	M and N are single digits between 1 and 9. This parameter means that the sequence of the Mth through the Nth words in the search criteria is inserted in place of %vM-N. For example, if the filter is (cn=%v1-2), entering the words Barbara Jensen results in the filter (cn=Barbara Jensen).

Table 6-1 Filter Parameters (*Continued*)

Parameter	Description
%vN-	<i>N</i> is a single digit between 1 and 9. This parameter means that the sequence of the <i>N</i> th through the last words in the search criteria is inserted in place of %vN-. For example, if the filter is (cn=%v2-), entering the words Ms. Barbara Jensen results in the filter (cn=Barbara Jensen).

Loading a Filter Configuration File

To load a filter configuration file, call the `ldap_init_getfilter()` function. You can also read the filter configuration file from a buffer in memory by calling the `ldap_init_getfilter_buf()` function. Both functions return a pointer to an `LDAPFiltDesc` structure, which contains information about the filter. If an error occurs, both functions return `NULL`.

Retrieving Filters

After loading a filter configuration file into memory, you can retrieve filters based on the search criteria. For example, if the search criteria is an e-mail address (`bjensen@example.com`), you can have your client automatically search for this value in the `mail` attribute of person entries.

To retrieve the first filter that matches the search criteria, call the `ldap_getfirstfilter()` function. To get the next filter that matches the search criteria, call the `ldap_getnextfilter()` function. Both functions return a pointer to an `LDAPFiltInfo` structure, which contains information about the filter. Code Example 6-3 retrieves filters that match the search criteria.

Code Example 6-3 Retrieving Configuration Filters

```
#include <stdio.h>
#include <ldap.h>
...
LDAP      *ld;
LDAPMessage *result, *e;
BerElement *ber;
char      *a, *dn;
char      **vals;
int i;
LDAPFiltDesc *ldfp;
LDAPFiltInfo *ldfi;
char buf[ 80 ]; /* contains the search criteria */
int found;
```

Code Example 6-3 Retrieving Configuration Filters (*Continued*)

```

/* Load the filter configuration file into an LDAPFilterDesc structure. */
if ( ( ldfp = ldap_init_getfilter( "myfilters.conf" ) ) == NULL ) {
    perror( "Cannot open filter configuration file" );
}

/* Select a filter to use when searching for the value in buf.
Use filters under the "people" tag in the filter configuration file. */
found = 0;
for ( ldfi = ldap_getfirstfilter( ldfp, "people", buf ); ldfi != NULL;
      ldfi = ldap_getnextfilter( ldfp ) ) {

    /* Use the selected filter to search the directory. */
    if ( ldap_search_s( ld, "dc=example,dc=com", ldfi->lfi_scope,
        ldfi->lfi_filter, NULL, 0, &result ) != LDAP_SUCCESS ) {
        ldap_perror( ld, "ldap_search_s" );
        return( 1 );
    } else {

        /* Once a filter gets results back, stop iterating through
        the different filters. */
        if ( ( found = ldap_count_entries( ld, result ) > 0 ) ) {
            break;
        } else {
            ldap_msgfree( result );
        }
    }
}

if ( found == 0 ) {
    printf( "No matching entries found.\n" );
} else {
    printf( "Found %d match%s where %s \"%s\"\n\n", found,
        found == 1 ? "" : "es", ldfi->lfi_desc, buf );
}

ldap_msgfree( result );
ldap_getfilter_free( ldfp );
...

```

Suppose that the search criteria is `bjensen@example.com` and the client application finds one matching entry. In this case, the application prints the following output:

Found 1 match where email address is `bjensen@example.com`

Filter Prefixes and Suffixes

If you need to apply a filter to all searches, you can add a filter prefix and suffix to all filters (rather than add the criteria to all filters). The prefix is automatically added to any filter retrieved through the `ldap_getfirstfilter()` and `ldap_getnextfilter()` functions as is the required suffix `[]` needed to balance the number of parentheses. For example, suppose you use this filter in a filter configuration file:

```
(cn=Babs Jensen)
```

If you retrieve this filter using `ldap_getfirstfilter()` or `ldap_getnextfilter()`, you would get a filter that constrains your client searches to person entries for the defined filter:

```
(&(objectClass=person)(cn=Babs Jensen))
```

To add a prefix and suffix automatically to all filters retrieved from the filter configuration file, call the `ldap_set_filter_additions()` function. Code Example 6-4 loads the filter configuration file named `myfilters.conf` into memory and adds the prefix `(&(objectClass=person))` and the suffix `[]` to each filter retrieved.

Code Example 6-4 Adding Prefixes and Suffixes

```
#include <ldap.h>
...
LDAPFiltDesc *lfdp;
char *filter_file = "myfilters.conf";
char *prefix = "&(objectClass=person)";
char *suffix = "[]";
...
lfdp = ldap_init_getfilter( filter_file );
ldap_setfilteraffixes( lfdp, prefix, suffix );
...
```

Freeing Filters from Memory

When you complete your search, you should free the `LDAPFiltDesc` structure from memory. To free `LDAPFiltDesc`, call the `ldap_getfilter_free()` function. Code Example 6-5 frees `LDAPFiltDesc` from memory after all searches are completed.

Code Example 6-5 Freeing Filters from Memory

```

#include <ldap.h>
...
LDAPFiltDesc *lfdp;
char *filter_file = "myfilters.conf";
...
/* Read the filter configuration file into an LDAPFiltDesc structure. */
lfdp = ldap_init_getfilter( filter_file );
...
/* Retrieve filters and perform searches. */
...
/* Free the configuration file (the LDAPFiltDesc structure). */
ldap_getfilter_free( lfdp );
...

```

Creating Filters Programmatically

You can build your own filters by calling the `ldap_create_filter()` function. Code Example 6-6 builds the filter (`mail=bjensen@example.com`).

Code Example 6-6 Creating Filters

```

char buf[LDAP_FILT_MAXSIZ];
char *pattern = "(%a=%v)";
char *attr = "mail";
char *value = "bjensen@example.com";
...
ldap_create_filter( buf, LDAP_FILT_MAXSIZ, pattern, NULL, NULL, attr,
    value, NULL );
...

```

Adding, Modifying and Deleting Entries

Sun™ Java System LDAP SDK for C contains functions to add, modify, delete, and rename entries in the directory. This chapter explains these functions and provides examples for calling them to perform their respective operations. It includes the following sections:

- Specifying Entry Information
- Adding an Entry
- Modifying an Entry
- Deleting an Entry
- Changing the DN of an Entry

Specifying Entry Information

In order to add or modify an entry in the directory, you need to specify information about the entry's attributes. In most cases, you will need to specify the following:

- The type of the attribute that you are working with; for example, the `sn` attribute or the `telephoneNumber` attribute.
- The values that you are adding or replacing in the attribute.
- The operation that you are performing when modifying an existing entry. (In other words, are you adding, modifying, or deleting the attribute in the existing entry?)

To specify this information, you use an `LDAPMod` structure as illustrated in Code Example 7-1.

Code Example 7-1 Defining a LDAPMod Structure

```
typedef struct ldapmod {
    int mod_op;
    char *mod_type;
    union {
        char **modv_strvals;
        struct berval **modv_bvals;
    } mod_vals;
#define mod_values      mod_vals.modv_strvals
#define mod_bvalues     mod_vals.modv_bvals
} LDAPMod;
```

Table 7-1 details the fields in the LDAPMod data structure.

Table 7-1 LDAPMod Field Descriptions

Field	Description
mod_op	The operation to be performed on the attribute and the type of data specified as the attribute values. This field can have one of the following values: • LDAP_MOD_ADD adds a value to the attribute. • LDAP_MOD_DELETE removes the value from the attribute. • LDAP_MOD_REPLACE replaces all existing values of the attribute. In addition, if you are specifying binary values in the mod_bvalues field, you should use the bitwise OR operator () to combine LDAP_MOD_BVALUES with the operation type. For example: <pre>mod->mod_op = LDAP_MOD_ADD LDAP_MOD_BVALUES</pre> If you are using the structure to add a new entry, you can specify 0 for the mod_op field (unless you are adding binary values and need to specify LDAP_MOD_BVALUES). See “Adding an Entry” on page 149 for details.
mod_type	Attribute type that you want to add, delete, or replace the values of (for example, sn or telephoneNumber).
mod_values	Pointer to a NULL-terminated array of string values for the attribute.
mod_bvalues	Pointer to a NULL-terminated array of berval structures for the attribute.

Note the following caveats regarding entry modifications and the parameters detailed in Table 7-1:

- If you specify `LDAP_MOD_DELETE` in the `mod_op` field and you remove all values in an attribute, the attribute is removed from the entry.
- If you specify `LDAP_MOD_DELETE` in the `mod_op` field and `NULL` in the `mod_values` field, the attribute is removed from the entry.
- If you specify `LDAP_MOD_REPLACE` in the `mod_op` field and `NULL` in the `mod_values` field, the attribute is removed from the entry.
- If you specify `LDAP_MOD_REPLACE` in the `mod_op` field and the attribute does not exist in the entry, the attribute is added to the entry.
- If you specify `LDAP_MOD_ADD` in the `mod_op` field and the attribute does not exist in the entry, the attribute is added to the entry.

If you've allocated memory for the structures yourself, you should free the structures when you're finished by calling the `ldap_mods_free()` function.

Adding an Entry

This is the procedure to add a new entry to the directory.

1. Use the `LDAPMod` structure to specify the name and values of each attribute.
2. Create an array of `LDAPMod` structures to represent the attributes in the entry.
3. Call `ldap_add_ext()` or `ldap_add_ext_s()`, passing in the array of `LDAPMod` structures and a distinguished name (DN) for the entry.
4. Call the `ldap_mods_free()` function to free any `LDAPMod` structures you've allocated.

Specifying Values for Attributes

There are three ways to specify a value for an attribute. You can specify a single value, multiple values, or you can add binary data as the value of an attribute.

Specifying a Single Value in an Attribute

To specify a value in an attribute, set the `mod_op`, `mod_type`, and `mod_values` fields in an `LDAPMod` structure. Code Example 7-2 sets up the structure for the `sn` attribute.

Code Example 7-2 Setting Up an Attribute Structure

```
#include "ldap.h"
...
LDAPMod attribute1;
char *sn_values[] = { "Jensen", NULL };
...
attribute1.mod_op = 0;
attribute1.mod_type = "sn";
attribute1.mod_values = sn_values;
...
```

Because you are specifying an attribute for a new entry (rather than for an existing entry), you can set the `mod_op` field to 0. For an existing entry, the `mod_op` field identifies the type of change you are making to the entry.

Specifying Multiple Values in an Attribute

If an attribute has more than one value, specify the values in the `mod_values` array. Code Example 7-3 specifies two values for the `cn` attribute.

Code Example 7-3 Specifying Multiple Values in an Attribute

```
#include "ldap.h"
...
LDAPMod attribute2, attribute3;
char *cn_values[] = { "Barbara Jensen", "Babs Jensen", NULL };
char *objectClass_values[] = { "top", "person",
    "organizationalPerson", "inetOrgPerson", NULL };
...
attribute2.mod_op = 0;
attribute2.mod_type = "cn";
attribute2.mod_values = cn_values;
attribute3.mod_op = 0;
attribute3.mod_type = "objectClass";
attribute3.mod_values = objectClass_values;
...
```

Specifying Binary Data as the Value of an Attribute

If the attribute contains binary data (rather than string values), specify the data in a `berval` structure like that in Code Example 7-4.

Code Example 7-4 Berval Structure

```

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

```

Table 7-2 lists the berval structure field descriptions.

Table 7-2 berval Field Descriptions

Field	Description
<code>bv_len</code>	The length of the data
<code>bv_val</code>	A pointer to the binary data

After creating the berval structures for the binary data, you need to:

1. Add the berval structures to the `mod_bvalues` field in the LDAPMod structure.
2. Use the bitwise OR operator (`|`) to combine the value of the `mod_op` field with `LDAP_MOD_BVALUES`. (When adding a new entry, you can just set the `mod_op` field to `LDAP_MOD_BVALUES`, since the `mod_op` field is 0 in this case.)

For example, suppose the file `my_photo.jpg` contains a JPEG photograph of Barbara Jensen. Code Example 7-5 sets the `jpegPhoto` attribute to the JPEG data of the photograph.

Code Example 7-5 Adding a Value to an Attribute

```

#include <stdio.h>
#include <sys/stat.h>
#include "ldap.h"
...
char *photo_data;
FILE *fp;
struct stat st;
LDAPMod attribute4;
struct berval photo_berval;
struct berval *jpegPhoto_values[2];
/* Get information about the JPEG file, including its size. */
if ( stat( "my_photo.jpg", &st ) != 0 ) {
    perror( "stat" );
    return( 1 );
}

```

Code Example 7-5 Adding a Value to an Attribute (*Continued*)

```

/* Open the JPEG file and read it into memory. */
if ( ( fp = fopen( "my_photo.jpg", "rb" ) ) == NULL ) {
    perror( "fopen" );
    return( 1 );
}
if ( ( ( photo_data = ( char * )malloc( st.st_size ) ) == NULL ) ||
    ( fread( photo_data, st.st_size, 1, fp ) != 1 ) ) {
    perror( photo_data ? "fread" : "malloc" );
    return( 1 );
}

fclose( fp );

attribute4.mod_op = LDAP_MOD_BVALUES;
attribute4.mod_type = "jpegPhoto";
photo_berval.bv_len = st.st_size;
photo_berval.bv_val = photo_data;
jpegPhoto_values[0] = &photo_berval;
jpegPhoto_values[1] = NULL;
attribute4.mod_values = jpegPhoto_values;

```

Specifying Attributes in the Entry

After Specifying Values for Attributes in LDAPMod structures, you need to construct an array of these structures. You will then pass a pointer to this array as a parameter to the function for creating a new entry.

NOTE Make sure that you create LDAPMod structures for all required attributes in the new entry. If using Sun Java System Directory Server, see the chapters on required object classes and attributes in the *Sun ONE Directory Server Administration Guide* (<http://docs.sun.com/doc/816-6698-10>). For information on which attributes are required in other LDAP servers, see your server's documentation.

Code Example 7-6 creates an array of LDAPMod structures.

Code Example 7-6 Adding Array of Structures to an Attribute

```

#include "ldap.h"
LDAPMod *list_of_mods[5]
LDAPMod attribute1, attribute2, attribute3, attribute4;
...
/* Code for filling the LDAPMod structures with values */
...

```


Code Example 7-6 Adding Array of Structures to an Attribute (*Continued*)

```
list_of_mods[0] = &attribute1;
list_of_mods[1] = &attribute2;
list_of_mods[2] = &attribute3;
list_of_mods[3] = &attribute4;
list_of_mods[4] = NULL;
...
```

Adding the Entry to the Directory

To add the entry to the directory, call one of the following:

- The synchronous `ldap_add_ext_s()` function.
- The asynchronous `ldap_add_ext()` function.

NOTE

If you've allocated `LDAPMod` structures yourself, you should free them when you are done. Call the `ldap_mods_free()` function to free `LDAPMod` structures.

Synchronous Add Operation

If you want to wait for the results of the add operation to complete before continuing, call the synchronous `ldap_add_ext_s()` function. This function sends an LDAP add request to the server and blocks other work until the server sends the results of the operation back to your client. The `ldap_add_ext_s()` function returns `LDAP_SUCCESS` if the operation completed successfully or an error code if a problem occurred.

NOTE

See the documentation on the `ldap_add_ext_s()` function for a list of the possible result codes.

Code Example 7-7 uses the synchronous `ldap_add_ext_s()` function to add the user *William Jensen* to the directory.

Code Example 7-7 Synchronous Add Operation

```
#include <stdio.h>
#include "ldap.h"
...
#define NEW_DN "uid=wbjensen,ou=People,dc=example,dc=com"
...
```

Code Example 7-7 Synchronous Add Operation (*Continued*)

```

LDAP          *ld;
LDAPMod       **mods;
char          *matched_msg = NULL, *error_msg = NULL;
int           rc;
...
/* Perform the add operation. */
rc = ldap_add_ext_s( ld, NEW_DN, mods, NULL, NULL );
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_add_ext_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
        fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
        fprintf( stderr,
            "Part of the DN that matches an existing entry: %s\n",
            matched_msg );
    }
} else {
    printf( "%s added successfully.\n", NEW_DN );
}
...

```

Code Example 7-8 is a sample program that calls the synchronous `ldap_add_ext_s()` function to add a new user to the directory.

Code Example 7-8 Sample Synchronous Add Entry Program

```

#include <stdio.h>
#include "ldap.h"
/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
#define BIND_DN "cn=Directory Manager"
#define BIND_PW "23skidoo"
#define NEW_DN "uid=wbjensen,ou=People,dc=example,dc=com"
#define NUM_MODS 5
int
main( int argc, char **argv )
{
    LDAP          *ld;
    LDAPMod       **mods;
    char          *matched_msg = NULL, *error_msg = NULL;
    int           i, rc;
    char *object_vals[] = { "top", "person", "organizationalPerson",
        "inetOrgPerson", NULL };

    char *cn_vals[] = { "William B Jensen", "William Jensen", "Bill Jensen",
        NULL };

```

Code Example 7-8 Sample Synchronous Add Entry Program (*Continued*)

```

char *sn_vals[] = { "Jensen", NULL };
char *givenname_vals[] = { "William", "Bill", NULL };
char *telephonenumber_vals[] = { "+1 415 555 1212", NULL };
/* Get a handle to an LDAP connection. */
if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
}
/* Bind to the server as the Directory Manager. */
rc = ldap_simple_bind_s( ld, BIND_DN, BIND_PW );
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
        fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
        fprintf( stderr,
            "Part of the DN that matches an existing entry: %s\n",
            matched_msg );
    }
    ldap_unbind_s( ld );
    return( 1 );
}
/* Construct the array of LDAPMod structures representing the attributes
of the new entry. */
mods = ( LDAPMod ** ) malloc( ( NUM_MODS + 1 ) * sizeof( LDAPMod * ) );
if ( mods == NULL ) {
    fprintf( stderr, "Cannot allocate memory for mods array\n" );
    exit( 1 );
}
for ( i = 0; i < NUM_MODS; i++ ) {
    if ( ( mods[ i ] = ( LDAPMod * ) malloc( sizeof( LDAPMod ) ) ) == NULL ) {
        fprintf( stderr, "Cannot allocate memory for mods element\n" );
        exit( 1 );
    }
}
mods[ 0 ]->mod_op = 0;
mods[ 0 ]->mod_type = "objectclass";
mods[ 0 ]->mod_values = object_vals;
mods[ 1 ]->mod_op = 0;
mods[ 1 ]->mod_type = "cn";
mods[ 1 ]->mod_values = cn_vals;
mods[ 2 ]->mod_op = 0;
mods[ 2 ]->mod_type = "sn";
mods[ 2 ]->mod_values = sn_vals;
mods[ 3 ]->mod_op = 0;
mods[ 3 ]->mod_type = "givenname";
mods[ 3 ]->mod_values = givenname_vals;
mods[ 4 ]->mod_op = 0;
mods[ 4 ]->mod_type = "telephonenumber";
mods[ 4 ]->mod_values = telephonenumber_vals;
mods[ 5 ] = NULL;
/* Perform the add operation. */
rc = ldap_add_ext_s( ld, NEW_DN, mods, NULL, NULL );

```

Code Example 7-8 Sample Synchronous Add Entry Program (*Continued*)

```

if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_add_ext_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL & *error_msg != '\0' ) {
        fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
        fprintf( stderr,
            "Part of the DN that matches an existing entry: %s\n",
            matched_msg );
    }
    } else {
        printf( "%s added successfully.\n", NEW_DN );
    }
    ldap_unbind_s( ld );
    for ( i = 0; i < NUM_MODS; i++ ) {
        free( mods[ i ] );
    }
    free( mods );
    return 0;
}

```

Asynchronous Add Operation

If you want to perform other work while waiting for the entry to be added, call the asynchronous `ldap_add_ext()` function. This function sends an LDAP add request to the server and returns an `LDAP_SUCCESS` result code if the request was successfully sent (or an LDAP result code if an error occurred).

The `ldap_add_ext()` function passes back a message ID identifying the add operation. To determine whether the server sent a response for this operation to your client, call the `ldap_result()` function and pass in this message ID. The `ldap_result()` function uses the message ID to determine if the server sent the results of the add operation. The function passes back the results in an `LDAPMessage` structure. You can call the `ldap_parse_result()` function to parse the `LDAPMessage` structure to determine if the operation was successful.

NOTE See the documentation on the `ldap_add_ext()` function for a list of the possible result codes.

Code Example 7-9 calls `ldap_add_ext()` to add the user *William Jensen* to the directory.

Code Example 7-9 Asynchronous Add Operation

```

#include <stdio.h>
#include "ldap.h"
...
#define NEW_DN "uid=wbjensen,ou=People,dc=example,dc=com"

...
LDAP          *ld;
LDAPMessage   *res;
LDAPControl   **serverctrls;
char          *matched_msg = NULL, *error_msg = NULL;
char          **referrals;
int           i, rc, parse_rc, msgid, finished = 0;

/* Timeout period for the ldap_result() function to wait for results. */
struct timeval zerotime;
zerotime.tv_sec = zerotime.tv_usec = 0L;
...
/* Send the LDAP add request. */
rc = ldap_add_ext( ld, NEW_DN, mods, NULL, NULL, &msgid );
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_add_ext: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    return( 1 );
}
...
/* Poll the server for the results of the add operation. */
while ( !finished ) {
    rc = ldap_result( ld, msgid, 0, &zerotime, &res );
    switch ( rc ) {
        case -1:
            /* An error occurred. */
            rc = ldap_get_lderrno( ld, NULL, NULL );
            fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
            ldap_unbind( ld );
            return( 1 );
        case 0:
            /* The timeout period specified by zerotime was exceeded,
               so call ldap_result() again and continue polling. */
            break;
        default:
            /* The function has retrieved the results of the add operation. */
            finished = 1;

            /* Parse the result to determine the result of
               the add operation. */
            parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg,
                &error_msg, &referrals, &serverctrls, 1 );
            if ( parse_rc != LDAP_SUCCESS ) {
                fprintf( stderr, "ldap_parse_result: %s\n",
                    ldap_err2string( parse_rc ) );
                ldap_unbind( ld );
                return( 1 );
            }
    }
}

```

Code Example 7-9 Asynchronous Add Operation (*Continued*)

```

/* Check the results of the LDAP add operation. */
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_add_ext: %s\n", ldap_err2string( rc ) );
    if ( error_msg != NULL & *error_msg != '\0' ) {
        fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
        fprintf( stderr,
            "Part of the DN that matches an existing entry: %s\n",
            matched_msg );
    }
} else {
    printf( "%s added successfully.\n", NEW_DN );
}
}
}
...

```

Code Example 7-10 is a sample program that calls the asynchronous `ldap_add_ext()` function to add a new user to the directory.

Code Example 7-10 Sample Asynchronous Add Program

```

#include <stdio.h>
#include "ldap.h"
void do_other_work();
int global_counter = 0;
void free_mods( LDAPMod **mods );
/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
#define BIND_DN "cn=Directory Manager"
#define BIND_PW "23skidoo"
#define NEW_DN "uid=wbjensen,ou=People,dc=example,dc=com"
#define NUM_MODS 5
int
main( int argc, char **argv )
{
    LDAP      *ld;
    LDAPMessage *res;
    LDAPMod    **mods;
    LDAPControl **serverctrls;
    char        *matched_msg = NULL, *error_msg = NULL;
    char        **referrals;
    int         i, rc, parse_rc, msgid, finished = 0;
    struct timeval zerotime;
    char *object_vals[] = { "top", "person", "organizationalPerson",
        "inetOrgPerson", NULL };

```

Code Example 7-10 Sample Asynchronous Add Program (*Continued*)

```

char *cn_vals[] = { "William B Jensen", "William Jensen", "Bill Jensen",
NULL };
char *sn_vals[] = { "Jensen", NULL };
char *givenname_vals[] = { "William", "Bill", NULL };
char *telephonenumber_vals[] = { "+1 415 555 1212", NULL };
zerotime.tv_sec = zerotime.tv_usec = 0L;
/* Get a handle to an LDAP connection. */
if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
perror( "ldap_init" );
return( 1 );
}
/* Bind to the server as the Directory Manager. */
rc = ldap_simple_bind_s( ld, BIND_DN, BIND_PW );
if ( rc != LDAP_SUCCESS ) {
fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
ldap_get_lderrno( ld, &matched_msg, &error_msg );
if ( error_msg != NULL && *error_msg != '\0' ) {
fprintf( stderr, "%s\n", error_msg );
}
if ( matched_msg != NULL && *matched_msg != '\0' ) {
fprintf( stderr,
"Part of the DN that matches an existing entry: %s\n",
matched_msg );
}
ldap_unbind_s( ld );
return( 1 );
}
/* Construct the array of LDAPMod structures representing the attributes
of the new entry. */
mods = ( LDAPMod ** ) malloc( ( NUM_MODS + 1 ) * sizeof( LDAPMod * ) );
if ( mods == NULL ) {
fprintf( stderr, "Cannot allocate memory for mods array\n" );
exit( 1 );
}
for ( i = 0; i < NUM_MODS; i++ ) {
if ( ( mods[ i ] = ( LDAPMod * ) malloc( sizeof( LDAPMod ) ) ) == NULL ) {
fprintf( stderr, "Cannot allocate memory for mods element\n" );
exit( 1 );
}
}
mods[ 0 ]->mod_op = 0;
mods[ 0 ]->mod_type = "objectclass";
mods[ 0 ]->mod_values = object_vals;
mods[ 1 ]->mod_op = 0;
mods[ 1 ]->mod_type = "cn";
mods[ 1 ]->mod_values = cn_vals;
mods[ 2 ]->mod_op = 0;
mods[ 2 ]->mod_type = "sn";
mods[ 2 ]->mod_values = sn_vals;
mods[ 3 ]->mod_op = 0;
mods[ 3 ]->mod_type = "givenname";
mods[ 3 ]->mod_values = givenname_vals;
mods[ 4 ]->mod_op = 0;
mods[ 4 ]->mod_type = "telephonenumber";
mods[ 4 ]->mod_values = telephonenumber_vals;

```

Code Example 7-10 Sample Asynchronous Add Program (*Continued*)

```

    mods[ 5 ] = NULL;
    /* Send the LDAP add request. */
    rc = ldap_add_ext( ld, NEW_DN, mods, NULL, NULL, &msgid );
    if ( rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_add_ext: %s\n", ldap_err2string( rc ) );
        ldap_unbind( ld );
        free_mods( mods );
        return( 1 );
    }
    /* Poll the server for the results of the add operation. */
    while ( !finished ) {
        rc = ldap_result( ld, msgid, 0, &zerotime, &res );
        switch ( rc ) {
            case -1:
                /* An error occurred. */
                rc = ldap_get_lderrno( ld, NULL, NULL );
                fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
                ldap_unbind( ld );
                free_mods( mods );
                return( 1 );
            case 0:
                /* The timeout period specified by zerotime was exceeded.
                 * This means that the server has still not yet sent the
                 * results of the add operation back to your client.
                 * Break out of this switch statement, and continue calling
                 * ldap_result() to poll for results. */
                break;
            default:
                /* The function has retrieved the results of the add operation
                 * from the server. */
                finished = 1;
                /* Parse the results received from the server. Note the last
                 * argument is a non-zero value, which indicates that the
                 * LDAPMessage structure will be freed when done. (No need
                 * to call ldap_msgfree().) */
                parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg, &error_msg,
                &referrals, &serverctrls, 1 );
                if ( parse_rc != LDAP_SUCCESS ) {
                    fprintf( stderr, "ldap_parse_result: %s\n", ldap_err2string(
                parse_rc ) );
                    ldap_unbind( ld );
                    free_mods( mods );
                    return( 1 );
                }
                /* Check the results of the LDAP add operation. */
                if ( rc != LDAP_SUCCESS ) {
                    fprintf( stderr, "ldap_add_ext: %s\n", ldap_err2string( rc ) );
                    if ( error_msg != NULL & *error_msg != '\0' ) {
                        fprintf( stderr, "%s\n", error_msg );
                    }
                }
                if ( matched_msg != NULL & *matched_msg != '\0' ) {
                    fprintf( stderr,
                        "Part of the DN that matches an existing entry: %s\n",
                        matched_msg );
                }
            }
        }
    }

```


Code Example 7-10 Sample Asynchronous Add Program (*Continued*)

```

    }
    } else {
        printf( "%s added successfully.\n"
               "Counted to %d while waiting for the add operation.\n",
               NEW_DN, global_counter );
    }
}
/* Do other work while waiting for the results of the add operation. */
if ( !finished ) {
    do_other_work();
}
}
ldap_unbind( ld );
free_mods( mods );
return 0;
}
/*
 * Free a mods array.
 */
void
free_mods( LDAPMod **mods )
{
    int i;
    for ( i = 0; i < NUM_MODS; i++ ) {
        free( mods[ i ] );
    }
    free( mods );
}

/*
 * Perform other work while polling for results.  This doesn't do anything
 * useful, but it could.
 */
void
do_other_work()
{
    global_counter++;
}

```

Modifying an Entry

This is the procedure to modify an entry.

1. Use the LDAPMod structure to specify a change to an attribute.
2. Create an array of LDAPMod structures representing the changes that need to be made.

3. Call the `ldap_modify_ext()` or `ldap_modify_ext_s()`, passing in the array of `LDAPMod` structures and the DN of the entry that you want modified.
4. Call the `ldap_mods_free()` function to free any `LDAPMod` structures you've allocated.

Types of Modifications

There are different types of modifications that can be performed on an entry. The following sections explain them.

Replacing the Values of an Attribute

To replace all existing values of an attribute, create an `LDAPMod` structure with the following:

- Set the `mod_type` field to the attribute type that you want to change (for example, `telephoneNumber`).
- Set the `mod_values` field to the new values of the attribute.
- Set the value of the `mod_op` field to `LDAP_MOD_REPLACE`.

If you want to specify binary data as `berval` structures (as opposed to string values), you need to do the following:

- Use the `mod_bvalues` field rather than the `mod_values` field.
- Use the bitwise OR operator (`|`) to combine the value `LDAP_MOD_BVALUES` with the value of the `mod_op` field.

Also take note of the following two caveats:

1. If you specify an attribute that does not exist in the entry, the attribute will be added to the entry.
2. If you set a `NULL` value for the attribute (either by setting the `mod_values` field to `NULL`, or by setting the `mod_bvalues` field to `NULL` when the `mod_op` field contains `LDAP_MOD_BVALUES`), the attribute will be removed from the entry.

Code Example 7-11 specifies a change that replaces the values of the `telephoneNumber` attribute.

Code Example 7-11 Modifying an Attribute Value

```
#include "ldap.h"
LDAPMod attribute1;
char *telephoneNumber_values[] = { "+1 650 555 1967", NULL };
attribute1.mod_type = "telephoneNumber";
attribute1.mod_op = LDAP_MOD_REPLACE;
attribute1.mod_values = telephoneNumber_values;
```

Removing Values from an Attribute

To remove values from an attribute, create an LDAPMod structure with the following:

- Set the `mod_type` field to the attribute type containing the values that you want to remove (for example, `facsimileTelephoneNumber`).
- Set the `mod_values` field to the values that you want removed from the attribute.
- Set the value of the `mod_op` field to `LDAP_MOD_DELETE`.

If you want to specify binary data as `berval` structures (as opposed to string values), you need to do the following:

- Use the `mod_bvalues` field rather than the `mod_values` field.
- Use the bitwise OR operator (`|`) to combine the value `LDAP_MOD_BVALUES` with the value of the `mod_op` field.

Also take note of the following two caveats:

1. If you remove all values from the attribute, the attribute will be removed from the entry.
2. If you set a NULL value for the attribute (either by setting the `mod_values` field to NULL, or by setting the `mod_bvalues` field to NULL when the `mod_op` field contains `LDAP_MOD_BVALUES`), the attribute will be removed from the entry.

Code Example 7-12 specifies the removal of one of the values of the `facsimileTelephoneNumber` attribute in the entry.

Code Example 7-12 Removing an Attribute Value

```
#include "ldap.h"
LDAPMod attribute2;
char *fax_values[] = { "+1 650 555 1967", NULL };
```

Code Example 7-12 Removing an Attribute Value (*Continued*)

```
attribute2.mod_type = "facsimileTelephoneNumber";
attribute2.mod_op = LDAP_MOD_DELETE;
attribute2.mod_values = fax_values;
...
```

Adding Values to an Attribute

To add values to an attribute to an entry, create an LDAPMod structure with the following:

- Set the `mod_type` field to the attribute type that you want to add values to (for example, `audio`).
- Set the `mod_values` field to the new values of the attribute.
- Set the value of the `mod_op` field to `LDAP_MOD_ADD`.

If the attribute contains binary data (as opposed to string values), you need to do the following:

- Use the `mod_bvalues` field rather than the `mod_values` field and put the values in `berval` structures.
- Use the bitwise OR operator (`|`) to combine the value `LDAP_MOD_BVALUES` with the value of the `mod_op` field.

NOTE If the attribute does not already exist in the entry, it will be added to the entry.

Code Example 7-13 adds values to the `audio` attribute of an entry.

Code Example 7-13 Adding Values to an Attribute

```
#include <stdio.h>
#include <sys/stat.h>
#include "ldap.h"
...
char *audio_data;
FILE *fp;
struct stat st;
LDAPMod attribute3;
struct berval audio_berval;
struct berval *audio_values[2];
...
/* Get information about the audio file, including its size. */
```

Code Example 7-13 Adding Values to an Attribute (*Continued*)

```

if ( stat( "my_sounds.au", &st ) != 0 ) {
    perror( "stat" );
    return( 1 );
}

/* Open the audio file and read it into memory. */
if ( ( fp = fopen( "my_sounds.au", "rb" ) ) == NULL ) {
    perror( "fopen" );
    return( 1 );
}

if ( ( ( audio_data = ( char * )malloc( st.st_size ) ) == NULL ) ||
      ( fread ( audio_data, st.st_size, 1, fp ) != 1 ) ) {
    perror( audio_data ? "fread" : "malloc" );
    return( 1 );
}

fclose( fp );
attribute3.mod_op = LDAP_MOD_ADD | LDAP_MOD_BVALUES;
attribute3.mod_type = "audio";
audio_berval.bv_len = st.st_size;
audio_berval.bv_val = audio_data;
audio_values[0] = &audio_berval;
audio_values[1] = NULL;
attribute3.mod_values = audio_values;
...

```

Removing an Attribute

You can remove an attribute from an entry in either of the following ways:

- Remove all values from the attribute.
- Set the `mod_op` field to `LDAP_MOD_REPLACE` or `LDAP_MOD_DELETE` and specify `NULL` for the `mod_values` field.

Adding an Attribute

If you add or replace values in an attribute that does not yet exist in the entry, the attribute will be added to the entry.

Creating an Array of Changes

After specifying the changes to attribute values in LDAPMod structures, you need to construct an array of these structures. (You will pass a pointer to this array as a parameter to the function for modifying the entry.) Code Example 7-14 creates an array of LDAPMod structures.

Code Example 7-14 Assembling an Array of LDAPMod Structures

```
#include "ldap.h"
...
LDAPMod *list_of_mods[4]
LDAPMod attribute1, attribute2, attribute3;
...
/* Code for filling the LDAPMod structures with values */
...
list_of_mods[0] = &attribute1;
list_of_mods[1] = &attribute2;
list_of_mods[2] = &attribute3;
list_of_mods[3] = NULL;
...
```

Modifying the Entry in the Directory

To modify the entry in the directory, call one of the following:

- The synchronous `ldap_modify_ext_s()` function.
- The asynchronous `ldap_modify_ext()` function.

Synchronous Modify Operation

If you want to wait for the results of the modify operation to complete before continuing, call the synchronous `ldap_modify_ext_s()` function. This function sends a modify request to the server and blocks all work until the server sends the results of the operation back to your client. It returns `LDAP_SUCCESS` if the operation completed successfully or an error code if a problem occurred.

NOTE See the documentation on the `ldap_modify_ext_s()` function for a list of the possible result codes.

Code Example 7-15 uses the synchronous `ldap_modify_ext_s()` function to modify the entry for the user *William Jensen* in the directory.

Code Example 7-15 Synchronous Modify Operation

```

#include <stdio.h>
#include "ldap.h"

#define MODIFY_DN "uid=wbjensen,ou=People,dc=example,dc=com"
...
LDAP      *ld;
LDAPMod    *mods[ 3 ];
char       *matched_msg = NULL, *error_msg = NULL;
int        rc;
...
/* Perform the modify operation. */
rc = ldap_modify_ext_s( ld, MODIFY_DN, mods, NULL, NULL );
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_modify_ext_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
        fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
        fprintf( stderr,
            "Part of the DN that matches an existing entry: %s\n",
            matched_msg );
    }
} else {
    printf( "%s modified successfully.\n", MODIFY_DN );
}
ldap_unbind_s( ld );
...

```

Code Example 7-16 is a sample program that calls the synchronous `ldap_modify_ext_s()` function to modify a user's entry in the directory. The program replaces the values of the mail attribute and adds a description attribute to the entry.

Code Example 7-16 Sample Synchronous Modify Program

```

#include <stdio.h>
#include <time.h>
#include "ldap.h"
/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
#define BIND_DN "cn=Directory Manager"
#define BIND_PW "23skidoo"
#define MODIFY_DN "uid=wbjensen,ou=People,dc=example,dc=com"
int
main( int argc, char **argv )

```

Code Example 7-16 Sample Synchronous Modify Program (*Continued*)

```

{
    LDAP          *ld;
    LDAPMod        mod0, mod1;
    LDAPMod        *mods[ 3 ];
    char          *matched_msg = NULL, *error_msg = NULL;
    char          *vals0[ 2 ], *vals1[ 2 ];
    time_t        now;
    char          buf[ 128 ];
    int            rc;
    /* Get a handle to an LDAP connection. */
    if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
        perror( "ldap_init" );
        return( 1 );
    }
    /* Bind to the server as the Directory Manager. */
    rc = ldap_simple_bind_s( ld, BIND_DN, BIND_PW );
    if ( rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
        ldap_get_lderrno( ld, &matched_msg, &error_msg );
        if ( error_msg != NULL && *error_msg != '\0' ) {
            fprintf( stderr, "%s\n", error_msg );
        }
        if ( matched_msg != NULL && *matched_msg != '\0' ) {
            fprintf( stderr,
                "Part of the DN that matches an existing entry: %s\n",
                matched_msg );
        }
        ldap_unbind_s( ld );
        return( 1 );
    }
    /* Construct the array of LDAPMod structures representing the changes that
    you want to make to attributes in the entry. */
    /* Specify the first modification, which replaces all values of the
    mail attribute with the value "wbj@example.com". */
    mod0.mod_op = LDAP_MOD_REPLACE;
    mod0.mod_type = "mail";
    vals0[0] = "wbj@example.com";
    vals0[1] = NULL;
    mod0.mod_values = vals0;
    /* Specify the second modification, which adds a value to the description
    attribute. If this attribute does not yet exist, the attribute is added to
    the entry. */
    mod1.mod_op = LDAP_MOD_ADD;
    mod1.mod_type = "description";
    time( &now );
    sprintf( buf, "This entry was modified with the modattrs program on %s",
        ctime( &now ) );
    /* Get rid of \n which ctime put on the end of the time string */
    if ( buf[ strlen( buf ) - 1 ] == '\n' ) {
        buf[ strlen( buf ) - 1 ] = '\0';
    }
    vals1[ 0 ] = buf;
    vals1[ 1 ] = NULL;
    mod1.mod_values = vals1;
}

```


Code Example 7-16 Sample Synchronous Modify Program (*Continued*)

```

    mods[ 0 ] = &mod0;
    mods[ 1 ] = &mod1;
    mods[ 2 ] = NULL;
    /* Perform the modify operation. */
    rc = ldap_modify_ext_s( ld, MODIFY_DN, mods, NULL, NULL );
    if ( rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_modify_ext_s: %s\n", ldap_err2string( rc ) );
        ldap_get_lderrno( ld, &matched_msg, &error_msg );
        if ( error_msg != NULL && *error_msg != '\0' ) {
            fprintf( stderr, "%s\n", error_msg );
        }
        if ( matched_msg != NULL && *matched_msg != '\0' ) {
            fprintf( stderr,
                "Part of the DN that matches an existing entry: %s\n",
                matched_msg );
        }
    } else {
        printf( "%s modified successfully.\n", MODIFY_DN );
    }
    ldap_unbind_s( ld );
    return 0;
}

```

Asynchronous Modify Operation

If you want to perform other work (in parallel) while waiting for the entry to be modified, call the asynchronous `ldap_modify_ext()` function. This function sends a modify request to the server and returns an `LDAP_SUCCESS` result code if the request was successfully sent or an LDAP result code if an error occurred.

The `ldap_modify_ext()` function passes back a message ID identifying the modify operation. To determine whether the server sent a response for this operation to your client, call the `ldap_result()` function and pass in this message ID. The `ldap_result()` function uses the message ID to determine if the server sent the results of the modify operation. The function passes back the results in an `LDAPMessage` structure.

You can call the `ldap_parse_result()` function to parse the `LDAPMessage` structure to determine if the operation was successful.

NOTE See the documentation on the `ldap_modify_ext()` function for a list of the possible result codes.

Code Example 7-17 calls `ldap_modify_ext()` to modify the entry for the user *William Jensen* in the directory.

Code Example 7-17 Asynchronous Modify Operation

```

#include <stdio.h>
#include "ldap.h"
...
#define MODIFY_DN "uid=wbjensen,ou=People,dc=example,dc=com"
...
LDAP      *ld;
LDAPMessage *res;
LDAPMod    *mods[ 3 ];
LDAPControl **serverctrls;
char      *matched_msg = NULL, *error_msg = NULL;
char      **referrals;
int        rc, parse_rc, msgid, finished = 0;

/* Timeout period for the ldap_result() function to wait for results. */
struct timeval zerotime;
zerotime.tv_sec = zerotime.tv_usec = 0L;
...
/* Send the LDAP modify request. */
rc = ldap_modify_ext( ld, MODIFY_DN, mods, NULL, NULL, &msgid );
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_modify_ext: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    return( 1 );
}

/* Poll the server for the results of the modify operation. */
while ( !finished ) {
    rc = ldap_result( ld, msgid, 0, &zerotime, &res );
    switch ( rc ) {
        case -1:
            /* An error occurred. */
            rc = ldap_get_lderrno( ld, NULL, NULL );
            fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
            ldap_unbind( ld );
            return( 1 );
        case 0:
            /* The timeout period specified by zerotime was exceeded,
               so call ldap_result() again and continue polling. */
            break;
        default:
            /* The function has retrieved the results of the
               modify operation. */
            finished = 1;

            /* Parse the results received from the server. */
            parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg,
                &error_msg, &referrals, &serverctrls, 1 );
            if ( parse_rc != LDAP_SUCCESS ) {
                fprintf( stderr, "ldap_parse_result: %s\n",
                    ldap_err2string( parse_rc ) );
                ldap_unbind( ld );
                return( 1 );
            }
    }
}

```

Code Example 7-17 Asynchronous Modify Operation (*Continued*)

```

/* Check the results of the LDAP modify operation. */
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_modify_ext: %s\n",
        ldap_err2string( rc ) );
    if ( error_msg != NULL & *error_msg != '\0' ) {
        fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
        fprintf( stderr,
            "Part of the DN that matches an existing entry: %s\n",
            matched_msg );
    }
} else {
    printf( "%s modified successfully.\n", MODIFY_DN );
}
}
}
ldap_unbind( ld );
...

```

Code Example 7-18 is a sample program that calls the asynchronous `ldap_modify_ext()` function to modify a user's entry in the directory. The program replaces the values of the mail attribute and adds a description attribute to the entry.

Code Example 7-18 Sample Asynchronous Modify Program

```

#include <stdio.h>
#include <time.h>
#include "ldap.h"
void do_other_work();
int global_counter = 0;
/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
#define BIND_DN "cn=Directory Manager"
#define BIND_PW "23skidoo"
#define MODIFY_DN "uid=wbjensen,ou=People,dc=example,dc=com"
int
main( int argc, char **argv )
{
    LDAP          *ld;
    LDAPMessage    *res;
    LDAPMod        mod0, mod1;
    LDAPMod        *mods[ 3 ];
    LDAPControl    **serverctrls;
    char           *matched_msg = NULL, *error_msg = NULL;
    char           *referrals;
    char           *vals0[ 2 ], *vals1[ 2 ];

```

Code Example 7-18 Sample Asynchronous Modify Program (*Continued*)

```

time_t      now;
char        buf[ 128 ];
int         rc, parse_rc, msgid, finished = 0;
struct timeval zerotime;
zerotime.tv_sec = zerotime.tv_usec = 0L;
/* Get a handle to an LDAP connection. */
if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
}
/* Bind to the server as the Directory Manager. */
rc = ldap_simple_bind_s( ld, BIND_DN, BIND_PW );
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
        fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
        fprintf( stderr,
            "Part of the DN that matches an existing entry: %s\n",
            matched_msg );
    }
    ldap_unbind_s( ld );
    return( 1 );
}
/* Construct the array of LDAPMod structures representing the changes that
you want to make to attributes in the entry. */
/* Specify the first modification, which replaces all values of the
mail attribute with the value "wbj@example.com". */
mod0.mod_op = LDAP_MOD_REPLACE;
mod0.mod_type = "mail";
vals0[0] = "wbj@example.com";
vals0[1] = NULL;
mod0.mod_values = vals0;
/* Specify the second modification, which adds a value to the description
attribute. If this attribute does not yet exist, the attribute is added to
the entry. */
mod1.mod_op = LDAP_MOD_ADD;
mod1.mod_type = "description";
time( &now );
sprintf( buf, "This entry was modified with the modattrs program on %s",
ctime( &now ) );
/* Get rid of \n which ctime put on the end of the time string */
if ( buf[ strlen( buf ) - 1 ] == '\n' ) {
    buf[ strlen( buf ) - 1 ] = '\0';
}
vals1[ 0 ] = buf;
vals1[ 1 ] = NULL;
mod1.mod_values = vals1;
mods[ 0 ] = &mod0;
mods[ 1 ] = &mod1;
mods[ 2 ] = NULL;
/* Send the LDAP modify request. */

```

Code Example 7-18 Sample Asynchronous Modify Program (Continued)

```

rc = ldap_modify_ext( ld, MODIFY_DN, mods, NULL, NULL, &msgid );
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_modify_ext: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    return( 1 );
}
/* Poll the server for the results of the modify operation. */
while ( !finished ) {
    rc = ldap_result( ld, msgid, 0, &zerotime, &res );
    switch ( rc ) {
        case -1:
            /* An error occurred. */
            rc = ldap_get_lderrno( ld, NULL, NULL );
            fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
            ldap_unbind( ld );
            return( 1 );
        case 0:
            /* The timeout period specified by zerotime was exceeded.
               This means that the server has still not yet sent the
               results of the modify operation back to your client.
               Break out of this switch statement, and continue calling
               ldap_result() to poll for results. */
            break;
        default:
            /* The function has retrieved the results of the modify operation from
               the server. */
            finished = 1;
            /* Parse the results received from the server. Note the last
               argument is a non-zero value, which indicates that the
               LDAPMessage structure will be freed when done. (No need
               to call ldap_msgfree().) */
            parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg, &error_msg,
            &referrals, &serverctrls, 1 );
            if ( parse_rc != LDAP_SUCCESS ) {
                fprintf( stderr, "ldap_parse_result: %s\n", ldap_err2string(
                parse_rc ) );
                ldap_unbind( ld );
                return( 1 );
            }
            /* Check the results of the LDAP add operation. */
            if ( rc != LDAP_SUCCESS ) {
                fprintf( stderr, "ldap_modify_ext: %s\n", ldap_err2string( rc ) );
                if ( error_msg != NULL & *error_msg != '\0' ) {
                    fprintf( stderr, "%s\n", error_msg );
                }
                if ( matched_msg != NULL && *matched_msg != '\0' ) {
                    fprintf( stderr,
                        "Part of the DN that matches an existing entry: %s\n",
                        matched_msg );
                }
            }
            else {
                printf( "%s modified successfully.\n"
                    "Counted to %d while waiting for the modify operation.\n",
                    MODIFY_DN, global_counter );
            }
    }
}

```

Code Example 7-18 Sample Asynchronous Modify Program (*Continued*)

```

    }
}
/* Do other work while waiting for the results of the modify operation. */
if ( !finished ) {
    do_other_work();
}
}
ldap_unbind( ld );
return 0;
}

/*
 * Perform other work while polling for results. This doesn't do anything
 * useful, but it could.
 */
void
do_other_work()
{
    global_counter++;
}

```

Deleting an Entry

To remove an entry from the directory, call one of the following:

- The synchronous `ldap_delete_ext_s()` function.
- The asynchronous `ldap_delete_ext()` function.

Synchronous Delete Operation

If you want to wait for the results of the delete operation to complete before continuing, call the synchronous `ldap_delete_ext_s()` function. This function sends a delete request to the server and blocks all other processes until the server sends the results of the operation back to your client. The `ldap_delete_ext_s()` function returns `LDAP_SUCCESS` if the operation completed successfully or an error code if a problem occurred.

NOTE See the documentation on the `ldap_delete_ext_s()` function for a list of the possible result codes.

Code Example 7-19 uses the synchronous `ldap_delete_ext_s()` function to remove the entry for user *William Jensen* from the directory.

Code Example 7-19 Synchronous Delete Operation

```
#include <stdio.h>
#include "ldap.h"
...
#define DELETE_DN "uid=wjensen,ou=People,dc=example,dc=com"
...
LDAP      *ld;
char      *matched_msg = NULL, *error_msg = NULL;
int        rc;
...
/* Perform the delete operation. */
rc = ldap_delete_ext_s( ld, DELETE_DN, NULL, NULL );
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_delete_ext_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
        fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
        fprintf( stderr,
            "Part of the DN that matches an existing entry: %s\n",
            matched_msg );
    }
} else {
    printf( "%s deleted successfully.\n", DELETE_DN );
}
ldap_unbind_s( ld );
...
```

Code Example 7-20 is a sample program that calls the synchronous `ldap_delete_ext_s()` function to delete a user's entry from the directory.

Code Example 7-20 Sample Synchronous Delete Program

```
#include <stdio.h>
#include "ldap.h"
/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
#define BIND_DN "cn=Directory Manager"
#define BIND_PW "23skidoo"
#define DELETE_DN "uid=wjensen,ou=People,dc=example,dc=com"
int
main( int argc, char **argv )
{
```

Code Example 7-20 Sample Synchronous Delete Program (*Continued*)

```

LDAP      *ld;
char      *matched_msg = NULL, *error_msg = NULL;
int       rc;
/* Get a handle to an LDAP connection. */
if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
}
/* Bind to the server as the Directory Manager. */
rc = ldap_simple_bind_s( ld, BIND_DN, BIND_PW );
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
        fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
        fprintf( stderr,
            "Part of the DN that matches an existing entry: %s\n",
            matched_msg );
    }
    ldap_unbind_s( ld );
    return( 1 );
}
/* Perform the delete operation. */
rc = ldap_delete_ext_s( ld, DELETE_DN, NULL, NULL );
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_delete_ext_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
        fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
        fprintf( stderr,
            "Part of the DN that matches an existing entry: %s\n",
            matched_msg );
    }
} else {
    printf( "%s deleted successfully.\n", DELETE_DN );
}
ldap_unbind_s( ld );
return 0;
}

```

Asynchronous Delete Operation

If you want to perform other work (in parallel) while waiting for the entry to be deleted, call the asynchronous `ldap_delete_ext()` function. This function sends a delete request to the server and returns an `LDAP_SUCCESS` result code if the request was successfully sent (or an LDAP result code if an error occurred).

The `ldap_delete_ext()` function passes back a message ID identifying the delete operation. To determine whether the server sent a response for this operation to your client, call the `ldap_result()` function and pass in this message ID. The `ldap_result()` function uses the message ID to determine if the server sent the results of the delete operation. The function passes back the results in an `LDAPMessage` structure. You can call the `ldap_parse_result()` function to parse the `LDAPMessage` structure to determine if the operation was successful.

NOTE See the documentation on the `ldap_delete_ext()` function for a list of the possible result codes.

Code Example 7-21 calls `ldap_delete_ext()` to remove the user *William Jensen* from the directory.

Code Example 7-21 Asynchronous Delete Operation

```
#include <stdio.h>
#include "ldap.h"
...
#define DELETE_DN "uid=wjensen,ou=People,dc=example,dc=com"
...
LDAP      *ld;
LDAPMessage *res;
LDAPControl **serverctrls;
char      *matched_msg = NULL, *error_msg = NULL;
char      **referrals;
int        rc, parse_rc, msgid, finished = 0;

/* Timeout period for the ldap_result() function to wait for results. */
struct timeval zerotime;
zerotime.tv_sec = zerotime.tv_usec = 0L;
...
/* Send the LDAP delete request. */
rc = ldap_delete_ext( ld, DELETE_DN, NULL, NULL, &msgid );
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_delete_ext: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    return( 1 );
}

/* Poll the server for the results of the delete operation. */
while ( !finished ) {
    rc = ldap_result( ld, msgid, 0, &zerotime, &res );
    switch ( rc ) {
        case -1:
            /* An error occurred. */
            rc = ldap_get_lderrno( ld, NULL, NULL );
            fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
            ldap_unbind( ld );
            return( 1 );
    }
}
```

Code Example 7-21 Asynchronous Delete Operation (*Continued*)

```

case 0:
    /* The timeout period specified by zerotime was exceeded,
       so call ldap_result() again and continue polling. */
    break;
default:
    /* The function has retrieved the results of the
       delete operation. */
    finished = 1;

    /* Parse the results received from the server. */
    parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg,
                                &error_msg, &referrals, &serverctrls, 1 );
    if ( parse_rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_parse_result: %s\n",
                ldap_err2string( parse_rc ) );
        ldap_unbind( ld );
        return( 1 );
    }
    /* Check the results of the LDAP delete operation. */
    if ( rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_delete_ext: %s\n",
                ldap_err2string( rc ) );
        if ( error_msg != NULL & *error_msg != '\0' ) {
            fprintf( stderr, "%s\n", error_msg );
        }
        if ( matched_msg != NULL && *matched_msg != '\0' ) {
            fprintf( stderr,
                    "Part of the DN that matches an existing entry: %s\n",
                    matched_msg );
        }
    } else {
        printf( "%s deleted successfully.\n", DELETE_DN );
    }
}
}
ldap_unbind( ld );
...

```

Code Example 7-22 is a sample program that calls the asynchronous `ldap_delete_ext()` function to delete a user's entry from the directory.

Code Example 7-22 Sample Asynchronous Delete Program

```

#include <stdio.h>
#include "ldap.h"
void do_other_work();
int global_counter = 0;
/* Change these as needed. */
#define HOSTNAME "localhost"

```

Code Example 7-22 Sample Asynchronous Delete Program (*Continued*)

```

#define PORTNUMBER LDAP_PORT
#define BIND_DN "cn=Directory Manager"
#define BIND_PW "23skidoo"
#define DELETE_DN "uid=wjensen,ou=People,dc=example,dc=com"
int
main( int argc, char **argv )
{
    LDAP      *ld;
    LDAPMessage *res;
    LDAPControl **serverctrls;
    char      *matched_msg = NULL, *error_msg = NULL;
    char      **referrals;
    int        rc, parse_rc, msgid, finished = 0;
    struct timeval zerotime;
    zerotime.tv_sec = zerotime.tv_usec = 0L;
    /* Get a handle to an LDAP connection. */
    if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
        perror( "ldap_init" );
        return( 1 );
    }
    /* Bind to the server as the Directory Manager. */
    rc = ldap_simple_bind_s( ld, BIND_DN, BIND_PW );
    if ( rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
        ldap_get_lderrno( ld, &matched_msg, &error_msg );
        if ( error_msg != NULL && *error_msg != '\0' ) {
            fprintf( stderr, "%s\n", error_msg );
        }
        if ( matched_msg != NULL && *matched_msg != '\0' ) {
            fprintf( stderr,
                "Part of the DN that matches an existing entry: %s\n",
                matched_msg );
        }
        ldap_unbind_s( ld );
        return( 1 );
    }
    /* Send the LDAP delete request. */
    rc = ldap_delete_ext( ld, DELETE_DN, NULL, NULL, &msgid );
    if ( rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_delete_ext: %s\n", ldap_err2string( rc ) );
        ldap_unbind( ld );
        return( 1 );
    }
    /* Poll the server for the results of the delete operation. */
    while ( !finished ) {
        rc = ldap_result( ld, msgid, 0, &zerotime, &res );
        switch ( rc ) {
            case -1:
                /* An error occurred. */
                rc = ldap_get_lderrno( ld, NULL, NULL );
                fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
                ldap_unbind( ld );
                return( 1 );
            case 0:
                /* The timeout period specified by zerotime was exceeded.

```

Code Example 7-22 Sample Asynchronous Delete Program (*Continued*)

```

        This means that the server has still not yet sent the
        results of the delete operation back to your client.
        Break out of this switch statement, and continue calling
        ldap_result() to poll for results. */
        break;
    default:
        /* The function has retrieved the results of the delete operation from the
        server. */
        finished = 1;
        /* Parse the results received from the server. Note the last
        argument is a non-zero value, which indicates that the
        LDAPMessage structure will be freed when done. (No need
        to call ldap_msgfree().) */
        parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg, &error_msg,
        &referrals, &serverctrls, 1 );
        if ( parse_rc != LDAP_SUCCESS ) {
            fprintf( stderr, "ldap_parse_result: %s\n", ldap_err2string(
            parse_rc ) );
            ldap_unbind( ld );
            return( 1 );
        }
        /* Check the results of the LDAP delete operation. */
        if ( rc != LDAP_SUCCESS ) {
            fprintf( stderr, "ldap_delete_ext: %s\n", ldap_err2string( rc ) );
            if ( error_msg != NULL & *error_msg != '\0' ) {
                fprintf( stderr, "%s\n", error_msg );
            }
            if ( matched_msg != NULL && *matched_msg != '\0' ) {
                fprintf( stderr,
                "Part of the DN that matches an existing entry: %s\n",
                matched_msg );
            }
        } else {
            printf( "%s deleted successfully.\n"
            "Counted to %d while waiting for the delete operation.\n",
            DELETE_DN, global_counter );
        }
    }
    /* Do other work while waiting for the results of the delete operation.
    */
    if ( !finished ) {
        do_other_work();
    }
    ldap_unbind( ld );
    return 0;
}
/*
 * Perform other work while polling for results. This doesn't do anything
 * useful, but it could.
 */
void
do_other_work()
{

```

Code Example 7-22 Sample Asynchronous Delete Program (*Continued*)

```

    global_counter++;
}

```

Changing the DN of an Entry

To change the distinguished name (DN) of an entry, call one of the following:

- The synchronous `ldap_rename_s()` function.
- The asynchronous `ldap_rename()` function.

For both functions, you can choose to delete the attribute that represents the old relative distinguished name (RDN). You can also change the location of the entry in the directory tree.

Synchronous Renaming Operation

If you want to wait for the results of the modify DN operation to complete before continuing, call the synchronous `ldap_rename_s()` function. This function sends a modify DN request to the server and blocks other work until the server sends the results of the operation back to your client. The `ldap_rename_s()` function returns `LDAP_SUCCESS` if the operation completed successfully or an error code if a problem occurred.

NOTE See the documentation on the `ldap_rename_s()` function for a list of the possible result codes.

Code Example 7-23 uses the synchronous `ldap_rename_s()` function to change the RDN of the entry for the user *William Jensen* in the directory.

Code Example 7-23 Synchronous Renaming Operation

```

#include <stdio.h>
#include "ldap.h"
...
#define OLD_DN "uid=wbjensen,ou=People,dc=example,dc=com"
#define NEW_RDN "uid=wjensen"
...
LDAP      *ld;

```

Code Example 7-23 Synchronous Renaming Operation (*Continued*)

```

char      *matched_msg = NULL, *error_msg = NULL;
int        rc;
...
/* Perform the modify DN operation. */
rc = ldap_rename_s( ld, OLD_DN, NEW_RDN, NULL, 1, NULL, NULL );
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_rename_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
        fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
        fprintf( stderr,
            "Part of the DN that matches an existing entry: %s\n",
            matched_msg );
    }
} else {
    printf( "%s renamed successfully.\n", OLD_DN );
}
ldap_unbind_s( ld );
...

```

Asynchronous Renaming Operation

If you want to perform other work (in parallel) while waiting for the entry to be renamed, call the asynchronous `ldap_rename()` function. This function sends a modify DN request to the server and returns an `LDAP_SUCCESS` result code if the request was successfully sent (or an LDAP result code if an error occurred).

The `ldap_rename()` function passes back a message ID identifying the modify DN operation. To determine whether the server sent a response for this operation to your client, call the `ldap_result()` function and pass in this message ID. `ldap_result()` uses the message ID to determine if the server sent the results of the modify DN operation. The function passes back the results in an `LDAPMessage` structure. You can call the `ldap_parse_result()` function to parse the `LDAPMessage` structure to determine if the operation was successful.

NOTE See the documentation on the `ldap_rename()` function for a list of the possible result codes.

Code Example 7-24 calls `ldap_rename()` to change the RDN of the user *William Jensen* in the directory.

Code Example 7-24 Asynchronous Renaming Operation

```

#include <stdio.h>
#include "ldap.h"
...
#define OLD_DN "uid=wbjensen,ou=People,dc=example,dc=com"
#define NEW_RDN "uid=wjensen"
...
LDAP      *ld;
LDAPMessage *res;
LDAPControl **serverctrls;
char      *matched_msg = NULL, *error_msg = NULL;
char      **referrals;
int        rc, parse_rc, msgid, finished = 0;

/* Timeout period for the ldap_result() function to wait for results. */
struct timeval zerotime;
zerotime.tv_sec = zerotime.tv_usec = 0L;
...
/* Send the LDAP modify DN request. */
rc = ldap_rename( ld, OLD_DN, NEW_RDN, NULL, 1, NULL, NULL, &msgid );
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_rename: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    return( 1 );
}

/* Poll the server for the results of the modify DN operation. */
while ( !finished ) {
    rc = ldap_result( ld, msgid, 0, &zerotime, &res );
    switch ( rc ) {
        case -1:
            /* An error occurred. */
            rc = ldap_get_lderrno( ld, NULL, NULL );
            fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
            ldap_unbind( ld );
            return( 1 );
        case 0:
            /* The timeout period specified by zerotime was exceeded,
               so call ldap_result() again and continue polling. */
            break;
        default:
            /* The function has retrieved the results of the
               modify DN operation. */
            finished = 1;

            /* Parse the results received from the server. */
            parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg,
                &error_msg, &referrals, &serverctrls, 1 );
            if ( parse_rc != LDAP_SUCCESS ) {
                fprintf( stderr, "ldap_parse_result: %s\n",
                    ldap_err2string( parse_rc ) );
                ldap_unbind( ld );
                return( 1 );
            }
    }
}

```

Code Example 7-24 Asynchronous Renaming Operation (*Continued*)

```

/* Check the results of the LDAP modify DN operation. */
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_rename: %s\n", ldap_err2string( rc ) );
    if ( error_msg != NULL & *error_msg != '\0' ) {
        fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL & *matched_msg != '\0' ) {
        fprintf( stderr,
            "Part of the DN that matches an existing entry: %s\n",
            matched_msg );
    }
} else {
    printf( "%s renamed successfully.\n", OLD_DN );
}
}
}
ldap_unbind( ld );
...

```

Deleting the Attribute from the Old RDN

Both `ldap_rename()` and `ldap_rename_s()` have a `deleteoldrdn` parameter that allows you to remove the old RDN from the entry. For example, suppose an entry has the following values for the `cn` attribute:

cn: Barbara Jensen

cn: Babs Jensen

Then:

```
ldap_modrdn2( "cn=Barbara Jensen", "cn=Barbie Jensen", 1 );
```

Is a function call that adds *Barbie Jensen* to this list of values and removes the *Barbara Jensen* value. The resulting entry has the following values:

cn: Barbie Jensen

cn: Babs Jensen

If a `0` is passed for the `deleteoldrdn` parameter (instead of a `1`):

```
ldap_modrdn2( "cn=Barbara Jensen", "cn=Barbie Jensen", 0 );
```

The *Barbara Jensen* value is not removed from the entry and the resulting entry has the following values:

cn: Barbie Jensen

cn: Babs Jensen

cn: Barbara Jensen

Changing the Location of the Entry

Both `ldap_rename()` and `ldap_rename_s()` have a `newparent` parameter that allows you to specify a new location for the entry in the directory tree. For example, if you pass `ou=Contractors,dc=example,dc=com` as the `newparent` parameter when renaming the entry `uid=bjensen,ou=People,dc=example,dc=com`, the entry is moved under `ou=Contractors,dc=example,dc=com` and the new DN for the entry is `uid=bjensen,ou=Contractors,dc=example,dc=com`.

CAUTION Not all LDAP servers support this feature. You might specify this argument and receive the LDAP result code `LDAP_UNWILLING_TO_PERFORM` with the error message "Server does not support moving of entries."

Synchronous Relocation of an Entry

Code Example 7-25 calls the synchronous `ldap_rename_s()` function to change the RDN of a user's entry in the directory.

Code Example 7-25 Synchronous Relocating of an Entry

```
#include <stdio.h>
#include "ldap.h"
/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
#define BIND_DN "cn=Directory Manager"
#define BIND_PW "23skidoo"
#define OLD_DN "uid=wbjensen,ou=People,dc=example,dc=com"
#define NEW_RDN "uid=wjensen"
int
main( int argc, char **argv )
{
    LDAP      *ld;
    char      *matched_msg = NULL, *error_msg = NULL;
    int       rc;
    /* Get a handle to an LDAP connection. */
    if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
        perror( "ldap_init" );
        return( 1 );
    }
    /* Bind to the server as the Directory Manager. */
    rc = ldap_simple_bind_s( ld, BIND_DN, BIND_PW );
    if ( rc != LDAP_SUCCESS ) {
```

Code Example 7-25 Synchronous Relocating of an Entry (*Continued*)

```

    fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
        fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
        fprintf( stderr,
            "Part of the DN that matches an existing entry: %s\n",
            matched_msg );
    }
    ldap_unbind_s( ld );
    return( 1 );
}
/* Perform the modify DN operation. */
rc = ldap_rename_s( ld, OLD_DN, NEW_RDN, NULL, 1, NULL, NULL );
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_rename_s: %s\n", ldap_err2string( rc ) );
    ldap_get_lderrno( ld, &matched_msg, &error_msg );
    if ( error_msg != NULL && *error_msg != '\0' ) {
        fprintf( stderr, "%s\n", error_msg );
    }
    if ( matched_msg != NULL && *matched_msg != '\0' ) {
        fprintf( stderr,
            "Part of the DN that matches an existing entry: %s\n",
            matched_msg );
    }
} else {
    printf( "%s renamed successfully.\n", OLD_DN );
}
ldap_unbind_s( ld );
return 0;
}

```

Asynchronous Relocation of an Entry

Code Example 7-26 calls the asynchronous `ldap_rename()` function to change the RDN of a user's entry in the directory.

Code Example 7-26 Asynchronous Relocating of an Entry

```

#include <stdio.h>
#include "ldap.h"
void do_other_work();
int global_counter = 0;
/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
#define BIND_DN "cn=Directory Manager"
#define BIND_PW "doug4444"

```

Code Example 7-26 Asynchronous Relocating of an Entry (*Continued*)

```

#define OLD_DN "uid=wbjensen,ou=People,dc=example,dc=com"
#define NEW_RDN "uid=wjensen"
int
main( int argc, char **argv )
{
    LDAP      *ld;
    LDAPMessage *res;
    LDAPControl **serverctrls;
    char      *matched_msg = NULL, *error_msg = NULL;
    char      **referrals;
    int        rc, parse_rc, msgid, finished = 0; struct timeval  zerotime;
    zerotime.tv_sec = zerotime.tv_usec = 0L;
    /* Get a handle to an LDAP connection. */
    if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
        perror( "ldap_init" );
        return( 1 );
    }
    /* Bind to the server as the Directory Manager. */
    rc = ldap_simple_bind_s( ld, BIND_DN, BIND_PW );
    if ( rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
        ldap_get_lderrno( ld, &matched_msg, &error_msg );
        if ( error_msg != NULL && *error_msg != '\0' ) {
            fprintf( stderr, "%s\n", error_msg );
        }
        if ( matched_msg != NULL && *matched_msg != '\0' ) {
            fprintf( stderr,
                "Part of the DN that matches an existing entry: %s\n",
                matched_msg );
        }
        ldap_unbind_s( ld );
        return( 1 );
    }
    /* Send the LDAP modify DN request. */
    rc = ldap_rename( ld, OLD_DN, NEW_RDN, NULL, 1, NULL, NULL, &msgid );
    if ( rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_rename: %s\n", ldap_err2string( rc ) );
        ldap_unbind( ld );
        return( 1 );
    }
    /* Poll the server for the results of the modify DN operation. */
    while ( !finished ) {
        rc = ldap_result( ld, msgid, 0, &zerotime, &res );
        switch ( rc ) {
            case -1:
                /* An error occurred. */
                rc = ldap_get_lderrno( ld, NULL, NULL );
                fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
                ldap_unbind( ld );
                return( 1 );
            case 0:
                /* The timeout period specified by zerotime was exceeded.
                   This means that the server has still not yet sent the
                   results of the modify DN operation back to your client.
                   Break out of this switch statement, and continue calling

```

Code Example 7-26 Asynchronous Relocating of an Entry (*Continued*)

```

        ldap_result() to poll for results. */
        break;
    default:
        /* The function has retrieved the results of the modify DN operation
           from the server. */
        finished = 1;
        /* Parse the results received from the server. Note the last
           argument is a non-zero value, which indicates that the
           LDAPMessage structure will be freed when done. (No need
           to call ldap_msgfree().) */
        parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg, &error_msg,
        &referrals, &serverctrls, 1 );
        if ( parse_rc != LDAP_SUCCESS ) {
            fprintf( stderr, "ldap_parse_result: %s\n", ldap_err2string(
        parse_rc ) );
            ldap_unbind( ld );
            return( 1 );
        }
        /* Check the results of the LDAP modify DN operation. */
        if ( rc != LDAP_SUCCESS ) {
            fprintf( stderr, "ldap_rename: %s\n", ldap_err2string( rc ) );
            if ( error_msg != NULL & *error_msg != '\0' ) {
                fprintf( stderr, "%s\n", error_msg );
            }
            if ( matched_msg != NULL && *matched_msg != '\0' ) {
                fprintf( stderr,
                "Part of the DN that matches an existing entry: %s\n",
                matched_msg );
            }
        } else {
            printf( "%s renamed successfully.\n"
                "Counted to %d while waiting for the modify DN operation.\n",
                OLD_DN, global_counter );
        }
    }
    /* Do other work while waiting for the results of the modify DN
       operation. */
    if ( !finished ) {
        do_other_work();
    }
    ldap_unbind( ld );
    return 0;
}
/*
 * Perform other work while polling for results. This doesn't do anything
 * useful, but it could.
 */
void
do_other_work()
{
    global_counter++;
}

```

Comparing Entries

This chapter explains how to use the Sun™ Java System LDAP SDK for C functions to compare data among entries in the directory. It includes the following sections:

- Overview
- Synchronous Comparison Operations
- Asynchronous Comparison Operations

Overview

The LDAP SDK for C has functions to determine if an attribute contains a certain string or binary value. To determine if an attribute contains a certain string value, call:

- The synchronous `ldap_compare_s()` function.
- The asynchronous `ldap_compare()` function.

To determine if an attribute contains a certain binary value, call:

- The synchronous `ldap_compare_ext_s()` function.
- The asynchronous `ldap_compare_ext()` function.

Synchronous Comparison Operations

If you want to wait for the results of the compare operation to complete before continuing, call the synchronous `ldap_compare_ext_s()` function to compare values in `berval` structures or the synchronous `ldap_compare_s()` function to compare string values. These functions send a compare request to the server and block work until the server sends the results of the operation back to your client. Both functions return one of the following values after the compare operation completes:

- `LDAP_COMPARE_TRUE` indicates that the attribute contains the specified value.
- `LDAP_COMPARE_FALSE` indicates that the attribute does not contain the specified value.

An error code indicates that a problem has occurred during the operation.

NOTE See the documentation on the `ldap_compare_ext_s()` function or the `ldap_compare_s()` function for a list of the possible result codes.

Code Example 8-1 uses the synchronous `ldap_compare_s()` function to determine if an entry has the value `bjensen@example.com` in the `mail` attribute.

Code Example 8-1 Synchronous Compare Operation

```
#include <stdio.h>
#include "ldap.h"
...
#define COMPARE_DN "uid=bjensen,ou=People,dc=example,dc=com"
#define COMPARE_ATTR "mail"
#define COMPARE_VALUE "bjensen@example.com"
...
LDAP      *ld;
char      *matched_msg = NULL, *error_msg = NULL;
int        rc;
...
/* Perform the compare operation. */
rc = ldap_compare_s( ld, COMPARE_DN, COMPARE_ATTR, COMPARE_VALUE );
switch( rc ) {
case LDAP_COMPARE_TRUE:
    printf( "%s has the value %s in the %s attribute.\n", COMPARE_DN,
            COMPARE_VALUE, COMPARE_ATTR );
    break;
case LDAP_COMPARE_FALSE:
    printf( "%s does not have the value %s in the %s attribute.\n",
            COMPARE_DN, COMPARE_VALUE, COMPARE_ATTR );
    break;
default:
```

Code Example 8-1 Synchronous Compare Operation (*Continued*)

```

fprintf( stderr, "ldap_compare_s: %s\n", ldap_err2string( rc ) );
ldap_get_lderrno( ld, &matched_msg, &error_msg );
if ( error_msg != NULL && *error_msg != '\0' ) {
    fprintf( stderr, "%s\n", error_msg );
}
if ( matched_msg != NULL && *matched_msg != '\0' ) {
    fprintf( stderr,
        "Part of the DN that matches an existing entry: %s\n",
        matched_msg );
}
break;
}
ldap_unbind_s( ld );
...

```

Code Example 8-2 is a sample program that calls the synchronous `ldap_compare_s()` function to determine if a user's entry has the value `bjensen@example.com` in the `mail` attribute.

Code Example 8-2 Sample Synchronous Comparison Program

```

#include <stdio.h>
#include "ldap.h"

/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
#define COMPARE_DN "uid=bjensen,ou=People,dc=example,dc=com"
#define COMPARE_ATTR "mail"
#define COMPARE_VALUE "bjensen@example.com"
int
main( int argc, char **argv )
{
    LDAP      *ld;
    char      *matched_msg = NULL, *error_msg = NULL;
    int       rc;
    /* Get a handle to an LDAP connection. */
    if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
        perror( "ldap_init" );
        return( 1 );
    }
    /* Bind anonymously to the server. */
    rc = ldap_simple_bind_s( ld, NULL, NULL );
    if ( rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
        ldap_get_lderrno( ld, &matched_msg, &error_msg );
        if ( error_msg != NULL && *error_msg != '\0' ) {
            fprintf( stderr, "%s\n", error_msg );
        }
    }
}

```

Code Example 8-2 Sample Synchronous Comparison Program (*Continued*)

```

        if ( matched_msg != NULL && *matched_msg != '\0' ) {
            fprintf( stderr,
                "Part of the DN that matches an existing entry: %s\n",
                matched_msg );
        }
        ldap_unbind_s( ld );
        return( 1 );
    }
    /* Perform the compare operation. */
    rc = ldap_compare_s( ld, COMPARE_DN, COMPARE_ATTR, COMPARE_VALUE );
    switch( rc ) {
        case LDAP_COMPARE_TRUE:
            printf( "%s has the value %s in the %s attribute.\n", COMPARE_DN,
                COMPARE_VALUE, COMPARE_ATTR );
            break;
        case LDAP_COMPARE_FALSE:
            printf( "%s does not have the value %s in the %s attribute.\n",
                COMPARE_DN, COMPARE_VALUE, COMPARE_ATTR );
            break;
        default:
            fprintf( stderr, "ldap_compare_s: %s\n", ldap_err2string( rc ) );
            ldap_get_lderrno( ld, &matched_msg, &error_msg );
            if ( error_msg != NULL && *error_msg != '\0' ) {
                fprintf( stderr, "%s\n", error_msg );
            }
            if ( matched_msg != NULL && *matched_msg != '\0' ) {
                fprintf( stderr,
                    "Part of the DN that matches an existing entry: %s\n",
                    matched_msg );
            }
            break;
    }
    ldap_unbind_s( ld );
    return 0;
}

```

Asynchronous Comparison Operations

If you want to perform other work (in parallel) while waiting for the comparison to complete, call the asynchronous `ldap_compare_ext()` function to compare values in berval structures or the asynchronous `ldap_compare()` function to compare string values. These functions send a compare request to the server and return an `LDAP_SUCCESS` result code if the request was successfully sent or an LDAP result code if an error occurred.

Both functions pass back a message ID identifying the compare operation. To determine whether the server sent a response for this operation to your client, call the `ldap_result()` function and pass in the message ID. `ldap_result()` uses the message ID to determine if the server sent the results of the compare operation. The function passes back the results in an `LDAPMessage` structure. You can call the `ldap_parse_result()` function to parse the `LDAPMessage` structure to determine if the operation was successful. The result code should be one of the following:

- `LDAP_COMPARE_TRUE` indicates that the attribute contains the specified value.
- `LDAP_COMPARE_FALSE` indicates that the attribute does not contain the specified value.

An error code indicates that a problem occurred during the operation.

NOTE See the documentation on the `ldap_compare_ext()` function or the `ldap_compare()` function for a list of the possible result codes.

Code Example 8-3 calls `ldap_compare()` to determine if an entry has the value `bjensen@example.com` in the `mail` attribute.

Code Example 8-3 Asynchronous Compare Operation

```
#include <stdio.h>
#include "ldap.h"
...
#define COMPARE_DN "uid=bjensen,ou=People,dc=example,dc=com"
#define COMPARE_ATTR "mail"
#define COMPARE_VALUE "bjensen@example.com"
...
LDAP      *ld;
LDAPMessage *res;
LDAPControl **serverctrls;
char      *matched_msg = NULL, *error_msg = NULL;
char      **referrals;
int        rc, parse_rc, msgid, finished = 0;
struct timeval zerotime;
zerotime.tv_sec = zerotime.tv_usec = 0L;
...
/* Send the LDAP compare request. */
msgid = ldap_compare( ld, COMPARE_DN, COMPARE_ATTR, COMPARE_VALUE );
if ( msgid < 0 ) {
    fprintf( stderr, "ldap_compare: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    return( 1 );
}
/* Poll the server for the results of the LDAP compare operation. */
while ( !finished ) {
    rc = ldap_result( ld, msgid, 0, &zerotime, &res );
```

Code Example 8-3 Asynchronous Compare Operation (*Continued*)

```

switch ( rc ) {
case -1:
    /* An error occurred. */
    rc = ldap_get_lderrno( ld, NULL, NULL );
    fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    return( 1 );

case 0:
    /* The timeout period specified by zerotime was exceeded, so
       call ldap_result() again and continue to poll for the
       results. */
    break;

default:
    /* The client has received the results of the
       LDAP compare operation from the server. */
    finished = 1;

    /* Parse the results received from the server.*/
    parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg,
        &error_msg, &referrals, &serverctrls, 1 );
    if ( parse_rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_parse_result: %s\n",
            ldap_err2string( parse_rc ) );
        ldap_unbind( ld );
        return( 1 );
    }

    /* Check the results of the LDAP compare operation. */
    switch ( rc ) {
    case LDAP_COMPARE_TRUE:
        printf( "%s has the value %s in the %s attribute.\n",
            COMPARE_DN, COMPARE_VALUE, COMPARE_ATTR );
        break;
    case LDAP_COMPARE_FALSE:
        printf( "%s does not have the value %s in the %s attribute.\n",
            COMPARE_DN, COMPARE_VALUE, COMPARE_ATTR );
        break;
    default:
        fprintf( stderr, "ldap_compare: %s\n", ldap_err2string( rc ) );
        if ( error_msg != NULL & *error_msg != '\0' ) {
            fprintf( stderr, "%s\n", error_msg );
        }
        if ( matched_msg != NULL && *matched_msg != '\0' ) {
            fprintf( stderr,
                "Part of the DN that matches an existing entry: %s\n",
                matched_msg );
        }
        break;
    }
}
}

```

Code Example 8-3 Asynchronous Compare Operation (*Continued*)

```
}
...
```

Code Example 8-4 is a sample program that calls the asynchronous `ldap_compare()` function to determine if a user's entry has the value `bjensen@example.com` in the mail attribute.

Code Example 8-4 Sample Asynchronous Comparison Program

```
#include <stdio.h>
#include "ldap.h"
void do_other_work();
int global_counter = 0;
/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER LDAP_PORT
#define COMPARE_DN "uid=bjensen,ou=People,dc=example,dc=com"
#define COMPARE_ATTR "mail"
#define COMPARE_VALUE "bjensen@example.com"
int
main( int argc, char **argv )
{
    LDAP      *ld;
    LDAPMessage *res;
    LDAPControl **serverctrls;
    char      *matched_msg = NULL, *error_msg = NULL;
    char      **referrals;
    int        rc, parse_rc, msgid, finished = 0;
    struct timeval zerotime;
    zerotime.tv_sec = zerotime.tv_usec = 0L;
    /* Get a handle to an LDAP connection. */
    if ( (ld = ldap_init( HOSTNAME, PORTNUMBER )) == NULL ) {
        perror( "ldap_init" );
        return( 1 );
    }
    /* Bind anonymously to the server. */
    rc = ldap_simple_bind_s( ld, NULL, NULL );
    if ( rc != LDAP_SUCCESS ) {
        fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
        ldap_get_lderrno( ld, NULL, &error_msg );
        if ( error_msg != NULL && *error_msg != '\0' ) {
            fprintf( stderr, "%s\n", error_msg );
        }
        ldap_unbind_s( ld );
        return( 1 );
    }
    /* Send the LDAP compare request. */
    msgid = ldap_compare( ld, COMPARE_DN, COMPARE_ATTR, COMPARE_VALUE );
    if ( msgid < 0 ) {
```

Code Example 8-4 Sample Asynchronous Comparison Program (*Continued*)

```

        fprintf( stderr, "ldap_compare: %s\n", ldap_err2string( rc ) );
        ldap_unbind( ld );
        return( 1 );
    }
    /* Poll the server for the results of the LDAP compare operation. */
    while ( !finished ) {
        rc = ldap_result( ld, msgid, 0, &zerotime, &res );
        switch ( rc ) {
            case -1:
                /* An error occurred. */
                rc = ldap_get_lderrno( ld, NULL, NULL );
                fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
                ldap_unbind( ld );
                return( 1 );
            case 0:
                /* The timeout period specified by zerotime was exceeded.
                 * This means that your client has not yet received the
                 * results of the LDAP compare operation.
                 * Break out of this switch statement, and continue calling
                 * ldap_result() to poll for the results. */
                break;
            default:
                /* The client has received the results of the
                 * LDAP compare operation from the server. */
                finished = 1;
                /* Parse the results received from the server. Note the last
                 * argument is a non-zero value, which indicates that the
                 * LDAPMessage structure will be freed when done. (No need
                 * to call ldap_msgfree().) */
                parse_rc = ldap_parse_result( ld, res, &rc, &matched_msg, &error_msg,
                &referrals, &serverctrls, 1 );
                if ( parse_rc != LDAP_SUCCESS ) {
                    fprintf( stderr, "ldap_parse_result: %s\n", ldap_err2string(
                parse_rc ) );
                    ldap_unbind( ld );
                    return( 1 );
                }
                /* Check the results of the LDAP compare operation. */
                switch ( rc ) {
                    case LDAP_COMPARE_TRUE:
                        printf( "%s has the value %s in the %s attribute.\n"
                            "Counted to %d while waiting for the compare operation.\n",
                            COMPARE_DN, COMPARE_VALUE, COMPARE_ATTR, global_counter );
                        break;
                    case LDAP_COMPARE_FALSE:
                        printf( "%s does not have the value %s in the %s attribute.\n"
                            "Counted to %d while waiting for the compare operation.\n",
                            COMPARE_DN, COMPARE_VALUE, COMPARE_ATTR, global_counter );
                        break;
                    default:
                        fprintf( stderr, "ldap_compare: %s\n", ldap_err2string( rc ) );
                        if ( error_msg != NULL & *error_msg != '\0' ) {
                            fprintf( stderr, "%s\n", error_msg );
                        }
                }
        }
    }

```

Code Example 8-4 Sample Asynchronous Comparison Program (*Continued*)

```

        if ( matched_msg != NULL && *matched_msg != '\0' ) {
            fprintf( stderr,
                "Part of the DN that matches an existing entry: %s\n",
                matched_msg );
        }
        break;
    }
}
/* Do other work while waiting for the results of the compare operation.
*/
    if ( !finished ) {
        do_other_work();
    }
}
ldap_unbind( ld );
return 0;
}
/*
 * Perform other work while polling for results.  This doesn't do anything
 * useful, but it could.
 */
void
do_other_work()
{
    global_counter++;
}

```


Working with LDAP URLs

The Sun™ Java System LDAP SDK for C contains functions to parse a Lightweight Directory Access Protocol (LDAP) URL into its components and process a search request specified by an LDAP URL. This chapter contains information on working with LDAP URLs. It contains the following sections:

- Understanding LDAP URLs
- Determining an LDAP URL
- Getting the Components of an LDAP URL
- Freeing the Components of an LDAP URL
- Processing an LDAP URL

Understanding LDAP URLs

An LDAP URL is a URL that begins with the `ldap://` or `ldaps://` prefix and specifies a search request sent to an LDAP server. The `ldap://` URL is used to connect to LDAP servers over unsecured connections, and the `ldaps://` URL is used to connect to LDAP servers over Secure Sockets Layer (SSL) connections. All LDAP URLs have the following syntax:

```
ldap[s]://hostname:port/base_dn?attributes?scope?filter
```

Table 9-1 details the components.

Table 9-1 Components of an LDAP URL

Component	Description
<i>hostname</i>	Name (or IP address in dotted format) of the LDAP server (for example, <code>ldap.server1.com</code> or <code>192.202.185.90</code>).

Table 9-1 Components of an LDAP URL (*Continued*)

Component	Description
<i>port</i>	Port number of the LDAP server (for example, 696). If no port is specified, the standard LDAP port 389 is used.
<i>base_dn</i>	DN of an entry in the directory. This DN identifies the entry that is starting point of the search. If this component is empty, the search starts at the root DN.
<i>attributes</i>	The attributes to be returned. To specify more than one attribute, use commas to delimit the attributes (for example, "cn,mail,telephoneNumber"). If no attributes are specified in the URL, all attributes are returned.
<i>scope</i>	The scope of the search, which can be one of these values: • base retrieves information only about the DN (<i>base_dn</i>) specified. • one retrieves information about entries one level below the DN (<i>base_dn</i>) specified. The base entry is not included in this scope. • sub retrieves information about entries at all levels below the DN (<i>base_dn</i>) specified. The base entry is included in this scope. If no scope is specified, the server performs a base search.
<i>filter</i>	Search filter to apply to entries within the specified scope of the search. If no filter is specified, the server uses the filter (objectClass=*).

Escaping Characters

Any *unsafe* characters in the URL need to be represented by a special sequence of characters (often called *escaping*). For example, a space must be represented as %20. Thus, the DN "ou=Product Development" must be encoded as "ou=Product%20Development".

Delimiting Fields

Attributes, scope, and filter are identified by their positions in the URL. If you do not want to specify any attributes, you still need to include question marks delimiting that field. For example, to specify a subtree search starting from dc=example,dc=com that returns all attributes for entries matching (sn=Jensen), use:

```
ldap://ldap.iplanet.com/dc=example,dc=com??sub?(sn=Jensen)
```

The two consecutive question marks (??) indicate that no attributes have been specified. Since no specific attributes are identified in the URL, all attributes are returned in the search.

NOTE The syntax for LDAP URLs does not include any means for specifying credentials or passwords. Search requests initiated through LDAP URLs are unauthenticated.

Examples of LDAP URLs

The following sections contain specific examples of LDAP URLs.

Specifying a Base Search

The first LDAP URL example specifies a base search for the entry with the DN `dc=example,dc=com`.

```
ldap://ldap.ipplanet.com/dc=example,dc=com
```

Note the following about this first example:

- Because no port number is specified, the standard LDAP port number (389) is used.
- Because no attributes are specified, the search returns all attributes.
- Because no search scope is specified, the search is restricted to the base entry `dc=example,dc=com`.
- Because no filter is specified, the default filter (`objectclass=*`) is used.

Retrieving One Attribute

The second example retrieves the `postalAddress` attribute of the `dc=example,dc=com` entry.

```
ldap://ldap.ipplanet.com/dc=example,dc=com?postalAddress
```

Note the following about this second example:

- Because no search scope is specified, the search is restricted to the base entry `dc=example,dc=com`.
- Because no filter is specified, the default filter (`objectclass=*`) is used.

Retrieving Multiple Attributes

This third example retrieves the `cn`, `mail`, and `telephoneNumber` attributes of the entry for *Barbara Jensen*.

```
ldap://ldap.ipplanet.com/uid=bjensen,ou=People,dc=example,dc=com? \
cn,mail,telephoneNumber
```

Note the following about this third example:

- Because no search scope is specified, the search is restricted to the base entry `uid=bjensen,ou=People,dc=example,dc=com`.
- Because no filter is specified, the default filter (`objectclass=*`) is used.

Retrieving Entries from Multiple Levels

This LDAP URL specifies a search for entries that have the last name *Jensen* and are at any level under `dc=example,dc=com`.

```
ldap://ldap.iplanet.com/dc=example,dc=com??sub?(sn=Jensen)
```

Note the following about this example:

- Because no attributes are specified, the search returns all attributes.
- Because the search scope is `sub`, the search encompasses the base entry `dc=example,dc=com` and entries at all levels under the base entry.

Retrieving Entries from One Level

This LDAP URL specifies a search for the object class for all entries one level under `dc=example,dc=com`.

```
ldap://ldap.iplanet.com/dc=example,dc=com?objectClass?one
```

- Because the search scope is `one`, the search encompasses all entries one level under the base entry `dc=example,dc=com`. (The search scope does not include the base entry.)
- Because no filter is specified, the default filter (`objectclass=*`) is used.

Determining an LDAP URL

To determine whether a URL is an LDAP URL, call the `ldap_is_ldap_url()` function. This function returns a nonzero value if the URL is an LDAP URL. If the URL is not an LDAP URL, the function returns 0. Code Example 9-1 determines if a URL is an LDAP URL.

Code Example 9-1 Determining an LDAP URL

```
#include <stdio.h>
#include <ldap.h>
...
char *my_url = "ldap://ldap.example.com/dc=example,dc=com";
```

Code Example 9-1 Determining an LDAP URL (*Continued*)

```

...
if ( ldap_is_ldap_url( my_url ) != 0 ) {
    printf( "%s is an LDAP URL.\n", my_url );
} else {
    printf( "%s is not an LDAP URL.\n", my_url );
}
...

```

`ldap_is_ldap_url()` determines whether a URL is an LDAP URL. To verify that an LDAP URL complies with the LDAP URL syntax, you should call the `ldap_url_parse()` function as detailed in “Getting the Components of an LDAP URL.”

Getting the Components of an LDAP URL

To retrieve the individual components of an LDAP URL, call `ldap_url_parse()`. This function returns the LDAP URL components in an `LDAPURLDesc` structure as shown in Code Example 9-2.

Code Example 9-2 LDAPURLDesc Structure

```

typedef struct ldap_url_desc {
    char *lud_host;
    int lud_port;
    char *lud_dn;
    char **lud_attrs;
    int lud_scope;
    char *lud_filter;
    unsigned long lud_options;
} LDAPURLDesc;

```

Table 9-2 is a list of the field descriptions.

Table 9-2 ldap_url_desc Field Descriptions

Field Name	Description
<code>lud_host</code>	The name of the host in the URL.
<code>lud_port</code>	The number of the port in the URL.

Table 9-2 ldap_url_desc Field Descriptions (*Continued*)

Field Name	Description
lud_dn	The distinguished name in the URL.
lud_attrs	A pointer to a NULL-terminated array of the attributes specified in the URL.
lud_scope	The scope of the search specified in the URL. This field can have the following values: - LDAP_SCOPE_BASE specifies a search of the base entry. - LDAP_SCOPE_ONELEVEL specifies a search of all entries one level under the base entry (not including the base entry). - LDAP_SCOPE_SUBTREE specifies a search of all entries at all levels under the base entry (including the base entry).
lud_filter	Search filter included in the URL.
lud_options	Options (if LDAP_URL_OPT_SECURE, indicates that the protocol is ldaps:// instead of ldap://).

Code Example 9-3 parses an LDAP URL and prints out each component of it.

Code Example 9-3 Parsing an LDAP URL

```
#include <stdio.h>
#include <ldap.h>
...
char *my_url =
"ldap://ldap.ipplanet.com:5000/dc=example,dc=com?cn,mail,telephoneNumber?sub
?
(sn=Jensen)";
LDAPURLDesc *ludpp;
int res, i;
...
if ( ( res = ldap_url_parse( my_url, &ludpp ) ) != 0 ) {
    switch( res ){
        case LDAP_URL_ERR_NOTLDAP:
            printf( "URL does not begin with \"ldap://\"\n" );
            break;
        case LDAP_URL_ERR_NODN:
            printf( "URL missing trailing slash after host or port\n" );
            break;
        case LDAP_URL_ERR_BADSCOPE:
            printf( "URL contains an invalid scope\n" );
            break;
        case LDAP_URL_ERR_MEM:
            printf( "Not enough memory\n" );
            break;
        default:
            printf( "Unknown error\n" );
    }
}
```

Code Example 9-3 Parsing an LDAP URL (*Continued*)

```

    }
    return( 1 );
}
printf( "Components of the URL:\n" );
printf( "Host name: %s\n", ludpp->lud_host );
printf( "Port number: %d\n", ludpp->lud_port );
if ( ludpp->lud_dn != NULL ) {
    printf( "Base entry: %s\n", ludpp->lud_dn );
} else {
    printf( "Base entry: Root DN\n" );
}
if ( ludpp->lud_attrs != NULL ) {
    printf( "Attributes returned: \n" );
    for ( i=0; ludpp->lud_attrs[i] != NULL; i++ ) {
        printf( "\t%s\n", ludpp->lud_attrs[i] );
    }
} else {
    printf( "No attributes returned.\n" );
}
printf( "Scope of the search: " );
switch( ludpp->lud_scope ) {
    case LDAP_SCOPE_BASE:
        printf( "base\n" );
        break;
    case LDAP_SCOPE_ONELEVEL:
        printf( "one\n" );
        break;
    case LDAP_SCOPE_SUBTREE:
        printf( "sub\n" );
        break;
    default:
        printf( "Unknown scope\n" );
}
printf( "Filter: %s\n", ludpp->lud_filter );
...

```

The code in Code Example 9-3 prints out the results pictured in Code Example 9-4.

Code Example 9-4 Results of Parsing an LDAP URL

```

Components of the URL:
Host name: ldap.netscape.com
Port number: 5000
Base entry: dc=example,dc=com
Attributes returned:
    cn
    mail
    telephoneNumber

```

Code Example 9-4 Results of Parsing an LDAP URL (*Continued*)

```
Scope of the search: sub
Filter: (sn=Jensen)
```

Freeing the Components of an LDAP URL

When you have finished working with the components of an LDAP URL, you should free the LDAPURLDesc structure from memory by calling the `ldap_free_urldesc()` function. Code Example 9-5 parses an LDAP URL and then frees the LDAPURLDesc structure from memory after verifying that the LDAP URL is valid.

Code Example 9-5 Freeing LDAPURLDesc Structure From Memory

```
#include <stdio.h>
#include <ldap.h>
...
char *my_url = "ldap://ldap.netscape.com:5000/dc=example,dc=com?cn,mail, \
telephoneNumber?sub?(sn=Jensen)";
LDAPURLDesc *ludpp;
int res, i;
...
if ( ( res = ldap_url_parse( my_url, &ludpp ) ) != 0 ) {
    switch( res ){
        case LDAP_URL_ERR_NOTLDAP:
            printf( "URL does not begin with \"ldap://\"\\n" );
            break;
        case LDAP_URL_ERR_NODN:
            printf( "URL does not contain a distinguished name\\n" );
            break;
        case LDAP_URL_ERR_BADSCOPE:
            printf( "URL contains an invalid scope\\n" );
            break;
        case LDAP_URL_ERR_MEM:
            printf( "Not enough memory\\n" );
            break;
        default:
            printf( "Unknown error\\n" );
    }
    return( 1 );
}
printf( "URL is a valid LDAP URL\\n" );
ldap_free_urldesc( ludpp );
...
```

Processing an LDAP URL

To process an LDAP URL search request, call either the `ldap_url_search_s()`, `ldap_url_search_st()`, or `ldap_url_search()` function.

- `ldap_url_search_s()` is a synchronous function that completes the search operation before returning. Call this function if you need to wait for the operation to finish before continuing other work. It returns `LDAP_SUCCESS` if the operation completed successfully. If an error occurred, the function returns an error code. (See Chapter 18, “Result Codes” for a complete listing of error codes.)
- `ldap_url_search_st()` is a synchronous function that allows a certain amount of time for the completion of the search operation. Call this function if you need to wait for the operation to complete and if you want to set a timeout period for the operation.
- `ldap_url_search()` is an asynchronous function that initiates the search operation but does not wait for the operation to complete. Call this function if you want to perform other work (in parallel) while waiting for the operation to complete. It returns a message ID identifying the search operation. To determine whether the operation is completed or still in progress, call the `ldap_result()` function.

After the operation is completed, call the `ldap_result2error()` function to determine if the operation was successful. If the operation completed successfully, the `ldap_result2error()` function returns `LDAP_SUCCESS`. If an error occurred, the function returns an error code. Code Example 9-6 processes a search request from an LDAP URL.

Code Example 9-6 Processing an LDAP URL Search Request

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result;
char *my_url = "ldap://ldap.netscape.com/dc=example,dc=com?cn,mail, \
telephoneNumber?sub?(sn=Jensen)";
/* Process the search request in the URL. */
if ( ldap_url_search_s( ld, my_url, 0, &result ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_url_search_s" );
    return( 1 );
}
```


Advanced Topics

Chapter 10, “Retrieving Server Information”

Chapter 11, “Connecting Over SSL”

Chapter 12, “Using SASL Authentication”

Chapter 13, “Working with LDAP Controls”

Chapter 15, “Extended Operations”

Chapter 14, “Writing Multithreaded Clients”

Retrieving Server Information

The Sun™ Java System LDAP SDK for C contains functions to access and modify information about your Lightweight Directory Access Protocol (LDAP) directory server. This chapter contains information about your server attributes and how to retrieve the information. It includes the following sections:

- Understanding DSEs
- Getting the Root DSE
- Determining LDAPv3 Support
- Getting Schema Information

Understanding DSEs

A directory server entry or *DSE* is an entry in the directory that contains information specific to the server. In a directory tree, the root of the tree is the root DSE. It is not part of any naming context. For example, it is above `dc=example,dc=com` in the directory tree.

NOTE	The root DSE is specified as part of the LDAPv3. LDAPv2 servers do not necessarily have a root DSE.

The root DSE can contain the following information:

- The naming contexts of this server (for example, `dc=example,dc=com`).
- URLs of alternate servers to contact if this server is unavailable.
- The versions of the LDAP supported by this server (for example, 2 or 3).

- The LDAPv3 controls supported by the server. See Chapter 13, “Working with LDAP Controls” for details.
- The SASL mechanisms supported by the server. See Chapter 12, “Using SASL Authentication” for details.
- The LDAPv3 extended operations supported by the server. See Chapter 15, “Extended Operations” for details.

Table 10-1 lists the types of information available in the different attributes of the root DSE.

Table 10-1 Information available in the root DSE

Attribute Name	Description of Values
namingContexts	The naming contexts supported by this server (for example, dc=example, dc=com).
altServer	LDAP URLs that identify other servers to contact if this server is unavailable.
supportedExtension	The object identifiers (OIDs) of the LDAPv3 extended operations supported by this server. If this attribute is not in the root DSE, the server does not support any extended operations.
supportedControl	The OIDs of the LDAPv3 controls supported by this server. If this attribute is not in the root DSE, the server does not support any LDAPv3 controls.
supportedSASLMechanisms	The names of the SASL mechanisms supported by the server. If this attribute is not in the root DSE, the server does not support any SASL mechanisms.
supportedLDAPVersion	The values of this attribute are the versions of the LDAP supported by this server (for example, 2 and 3).

Getting the Root DSE

This is the procedure to get the root DSE for an LDAP server.

1. Initialize an LDAP session by calling the `ldap_init()` function.
2. Turn off automatic referral handling by calling the `ldap_set_option()` function and setting the `LDAP_OPT_REFERRALS` option to `LDAP_OPT_OFF`.
3. Search the directory using the following criteria:
 - Set the search scope to a base search.

- Specify an empty string for the base DN.
- Use the search filter (objectclass=*).

NOTE For details on how to use API functions to search the directory, see Chapter 5, “Searching a LDAP Directory.”

4. Check the results of the search.

If the server returns a result code such as `LDAP_OPERATIONS_ERROR`, `LDAP_PROTOCOL_ERROR`, `LDAP_REFERRAL`, or `LDAP_NO_SUCH_OBJECT`, the LDAP server probably does not support LDAPv3.

Code Example 10-1 gets the root DSE for a server and prints out its attributes. The function assumes that you are passing in a valid connection handle (LDAP structure) that you have created by calling `ldap_init()`. The function returns 0 if successful or 1 if an error occurred.

Code Example 10-1 Getting the Root DSE Attribute Values

```
int printdse( LDAP *ld )
{
    int rc, i;
    char *matched_msg = NULL, *error_msg = NULL;
    LDAPMessage *result, *e;
    BerElement *ber;
    char *a;
    char **vals;
    char *attrs[3];
    /* Verify that the connection handle is valid. */
    if ( ld == NULL ) {
        fprintf( stderr, "Invalid connection handle.\n" );
        return( 1 );
    }
    /* Set automatic referral processing off. */
    if ( ldap_set_option( ld, LDAP_OPT_REFERRALS, LDAP_OPT_OFF ) != 0 ) {
        rc = ldap_get_lderrno( ld, NULL, NULL );
        fprintf( stderr, "ldap_set_option: %s\n", ldap_err2string( rc ) );
        return( 1 );
    }
    /* Search for the root DSE. */
    attrs[0] = "supportedControl";
    attrs[1] = "supportedExtension";
    attrs[2] = NULL;
    rc = ldap_search_ext_s( ld, "", LDAP_SCOPE_BASE, "(objectclass=*)", attrs,
        0, NULL, NULL, NULL, 0, &result );
    /* Check the search results. */
    switch( rc ) {
        /* If successful, the root DSE was found. */
        case LDAP_SUCCESS:
```

Code Example 10-1 Getting the Root DSE Attribute Values (*Continued*)

```

        break;
    /* If the root DSE was not found, the server does not comply
       with the LDAPv3 protocol. */
    case LDAP_PARTIAL_RESULTS:
    case LDAP_NO_SUCH_OBJECT:
    case LDAP_OPERATIONS_ERROR:
    case LDAP_PROTOCOL_ERROR:
        printf( "LDAP server returned result code %d (%s).\n",
                "This server does not support the LDAPv3 protocol.\n",
                rc, ldap_err2string( rc ) );
        return( 1 );
    /* If any other value is returned, an error must have occurred. */
    default:
        fprintf( stderr, "ldap_search_ext_s: %s\n", ldap_err2string( rc ) );
        return( 1 );
    }
    /* Since only one entry should have matched, get that entry. */
    e = ldap_first_entry( ld, result );
    if ( e == NULL ) {
        fprintf( stderr, "ldap_search_ext_s: Unable to get root DSE.\n");
        ldap_memfree( result );
        return( 1 );
    }

    /* Iterate through each attribute in the entry. */
    for ( a = ldap_first_attribute( ld, e, &ber );
          a != NULL; a = ldap_next_attribute( ld, e, ber ) ) {

        /* Print each value of the attribute. */
        if ((vals = ldap_get_values( ld, e, a)) != NULL ) {
            for ( i = 0; vals[i] != NULL; i++ ) {
                printf( "%s: %s\n", a, vals[i] );
            }

            /* Free memory allocated by ldap_get_values(). */
            ldap_value_free( vals );
        }

        /* Free memory allocated by ldap_first_attribute(). */
        ldap_memfree( a );
    }

    /* Free memory allocated by ldap_first_attribute(). */
    if ( ber != NULL ) {
        ber_free( ber, 0 );
    }

    printf( "\n" );
    /* Free memory allocated by ldap_search_ext_s(). */
    ldap_msgfree( result );
    ldap_unbind( ld );
    return( 0 );
}

```

Determining LDAPv3 Support

You can determine what version an LDAP server supports by getting the supportedLDAPVersion attribute from the root DSE. This attribute could contain the value 2 or 3.

NOTE You do not need to authenticate or bind before searching the directory. Unlike LDAPv2, LDAPv3 states that clients do not need to bind to the server before performing LDAP operations.

Code Example 10-2 connects to an LDAP server and determines whether or not that server supports the LDAPv3.

Code Example 10-2 Determining the Supported LDAP Version

```
/* Function for determining if the LDAP server supports LDAPv3.
   This function returns 1 if the server supports LDAPv3 or
   0 if the server does not support LDAPv3.
*/
int
check_version( char *hostname, int portnum )
{
    LDAP      *ld;
    int       i, rc, v3supported = 0;
    LDAPMessage *result, *e;
    BerElement *ber;
    LDAPControl **serverctrls = NULL, **clntctrls = NULL;
    char      *a, *dn;
    char      **vals;
    char      *attrs[2];
    char      *filter = "(objectClass=*)";
    /* Check arguments */
    if ( !hostname || !hostname[0] || !portnum ) {
        printf( "Error: hostname or port number not specified\n" );
        return( -1 );
    }
    /* Get a handle to an LDAP connection. */
    if ( (ld = ldap_init( hostname, portnum )) == NULL ) {
        perror( "ldap_init" );
        return( -1 );
    }
    /* Set automatic referral processing off. */
    if ( ldap_set_option(ld, LDAP_OPT_REFERRALS, LDAP_OPT_OFF) != LDAP_SUCCESS)
    {
        ldap_perror( ld, "ldap_set_option" );
    }
}
```

Code Example 10-2 Determining the Supported LDAP Version (*Continued*)

```

    return( -1 );
}
/* Search for the root DSE and get the supportedLDAPVersion attribute. */
attrs[0] = "supportedLDAPVersion";
attrs[1] = NULL;
rc = ldap_search_ext_s( ld, "", LDAP_SCOPE_BASE, filter, attrs, 0,
    serverctrls, clntctrls, NULL, 0, &result );
/* Check the search results. */
switch( rc ) {
    /* If successful, the root DSE was found. */
    case LDAP_SUCCESS:
        break;
    /* If the root DSE was not found, the server does not comply
       with the LDAPv3 protocol. */
    case LDAP_PARTIAL_RESULTS:
    case LDAP_NO_SUCH_OBJECT:
    case LDAP_OPERATIONS_ERROR:
    case LDAP_PROTOCOL_ERROR:
        ldap_perror( ld, "ldap_search_ext_s" );
        return( 0 );
        break;
    /* If an different result code is returned, an error may have
       occurred (for example, the server may be down. */
    default:
        ldap_perror( ld, "ldap_search_ext_s" );
        return( -1 );
        break;
}
/* Get the values of the supportedLDAPVersion attribute in the entry. */
if ( ( e = ldap_first_entry( ld, result ) ) != NULL &&
    ( a = ldap_first_attribute( ld, e, &ber ) ) != NULL &&
    ( vals = ldap_get_values( ld, e, a ) ) != NULL ) {
    for ( i = 0; vals[i] != NULL; i++ ) {
        if ( !strcmp( "3", vals[i] ) ) {
            v3supported = 1;
            break;
        }
    }
}
/* Free any memory allocated. */
ldap_value_free( vals );
ldap_memfree( a );
if ( ber != NULL ) {
    ber_free( ber, 0 );
}
}
/* Free memory allocated by ldap_search_ext_s(). */
ldap_msgfree( result );
/* Free the ld structure. */
ldap_unbind_s( ld );
/* Return a value indicating whether or not LDAPv3 is supported. */
return( v3supported );
}
...

```


Getting Schema Information

In the LDAPv3, an entry can specify the schema that defines the object classes, attributes, and matching rules used by the directory. This entry is called the subschema entry. To find the DN of the subschema entry, get the subschemaSubentry operational attribute from the root DSE or any entry.

NOTE	See “Specifying Attributes to Retrieve” on page 111 in Chapter 5, “Searching a LDAP Directory” for details on how to get attributes.

The subschema entry can have the following attributes:

- **objectClasses** specifies the object class definitions in the schema. Each value of this attribute is an object class that is known to the server.
- **attributeTypes** specifies the attribute type definitions in the schema. Each value of this attribute is an attribute type that is known to the server.
- **matchingRules** specifies the matching rule definitions in the schema. Each value of this attribute is a matching rule that is known to the server.
- **matchingRuleUse** specifies the use of a matching rule in the schema. (This specifies the attributes that can be used with this extensible matching rule.) Each value of this attribute is a matching rule use description.

For information on the the format of the attribute values, see *RFC 2252 - Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions* (<http://www.faqs.org/rfcs/rfc2252.html>).

Connecting Over SSL

The Sun™ Java System LDAP SDK for C includes functions to enable your application to connect to a Lightweight Directory Access Protocol (LDAP) server over a Secure Sockets Layer (SSL). This chapter covers the procedures for connecting and authenticating. It includes the following sections:

- SSL and the LDAP SDK for C
- Connecting to an LDAP Server Over SSL
- Handling Errors
- Installing Your Own SSL I/O Functions
- Using Certificate-Based Client Authentication

SSL and the LDAP SDK for C

The primary goal of the SSL protocol is to provide privacy and reliability between two communicating applications.

NOTE	SSL is not supported on all LDAP servers. The Sun Java System LDAP SDK for C only supports SSL 3.0 and does not support the Start Transport Layer Security (TLS) Operation. Also, SSL communication must take place on a separate TCP port. More information on SSL can be found in the <i>Secure Sockets Layer (SSL) Protocol (v3), Internet-Draft Revision 3</i> (http://wp.netscape.com/eng/ssl3/).

When a LDAP client connects to a LDAP server over SSL, the server identifies itself by sending a certificate to the client. The client then needs to determine whether or not the certificate authority (CA) that issued the certificate is trusted so it searches a certificate database for the certificate of the CA. If the client cannot find the certificate or if the certificate is marked *not trusted*, the client refuses to connect to

the server. The LDAP server may also request that the client send a certificate to authenticate itself. This part of the process is called *certificate-based client authentication*. If the client receives a request for a certificate from the server, the client retrieves its certificate from the certificate database and sends it to the server for authentication. After receiving the client's certificate, the LDAP server would determine whether or not the CA that issued the certificate is trusted. If the server cannot find the CA certificate or if the CA certificate is marked as *not trusted*, the server refuses to authenticate your client. If the CA is trusted, the server then uses the subject name in the certificate to determine if the client has access rights to perform the requested operation.

The LDAP SDK for C includes functions that allow you to connect an LDAP client to an LDAP server over SSL. The functions assume the following:

- Your client has access to a certificate database. The function call uses this certificate database to determine if it can trust the certificate sent from the server.

CAUTION Previous and later versions of these applications use different file formats for the certificate database. Attempting to use a different version of the certificate database will result in database errors.

- The database that you are using contains any one of the following:
 - a. The certificate of the CA that issued the server's certificate.
 - b. If the CAs are organized in a hierarchy, the certificate of any of the CAs in the hierarchy.
 - c. The certificate of the LDAP server.
- The CA certificate is marked as *trusted* in the certificate database.
- If you plan to use certificate-based client authentication, you also need:
 - a. A client certificate (issued by a CA trusted by the LDAP server) in the certificate database.
 - b. A public/private key pair in a Netscape key file.

The functions themselves allow you to:

- Set the session option for communicating with the server over SSL. For more information, see "Connecting to an LDAP Server Over SSL" on page 221.

- Replace the default I/O functions with your own I/O functions for communicating over SSL. For more information, see “Installing Your Own SSL I/O Functions” on page 223.
- Enable your client to send certificates to authenticate itself. For more information, see “Using Certificate-Based Client Authentication” on page 223.

Connecting to an LDAP Server Over SSL

To enable your LDAP client to connect to an LDAP server using SSL, you need to:

1. Initialize your client by calling one of the following functions:
 - a. Call `ldapssl_client_init()` if you do not plan to use certificate-based client authentication.
 - b. Call `ldapssl_clientauth_init()` if you plan to use certificate-based client authentication.
 - c. Call `ldapssl_clientauth_init()` if you plan to use certificate-based client authentication and need to specify either the name and path of the security module database or the method of verifying the server’s certificate.
2. Initialize an LDAP session with the secure server by calling the `ldapssl_init()` function. For an alternative way to accomplish this particular step, see “Alternative to `ldapssl_init()`” on page 222.

NOTE You need to initialize your client before initializing the LDAP session. The process of initializing the client opens the certificate database.

Code Example 11-1 initializes a client to connect to a secure LDAP server over SSL.

Code Example 11-1 Initializing a Client SSL Connection

```
if ( ldapssl_client_init( "/u/mozilla/.netscape/cert_file_name.db", NULL ) < 0 )
{
    printf( "Failed to initialize SSL client...\n" );
    return( 1 );
}
/* get a handle to an LDAP connection */
if ( (ld = ldapssl_init( "cert.netscape.com", LDAPS_PORT, 1 )) == NULL {
    perror( "ldapssl_init" );
    return( 1 );
}
...
```

Code Example 11-1 Initializing a Client SSL Connection (*Continued*)

```
/* Client can now perform LDAP operations on the secure LDAP server */
...
```

Alternative to ldapssl_init()

As an alternative to calling the `ldapssl_init()` function (Step 2 on page 221), you can do the following:

1. Initialize an LDAP session with the server by calling the standard initialization function `ldapssl_init()`.
2. Install the standard SSL I/O functions by calling `ldapssl_install_routines()`.
3. Set the SSL option in the LDAP structure by calling `ldap_set_option()`.

The effect of calling these three functions is the same as calling the `ldapssl_init()` function. Code Example 11-2 prepares a client to connect to a secure LDAP server over SSL using this alternate method.

Code Example 11-2 An Alternate Client SSL Initialization

```
if ( ldapssl_client_init( "/u/mozilla/.netscape/cert_file_name.db", NULL ) < 0 )
{
    printf( "Failed to initialize SSL client...\n" );
    return( 1 );
}
/* Initialize LDAP session */
if ( ( ld = ldap_init( MY_HOST, LDAPS_PORT ) ) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
}

/* Load SSL routines */
if ( ldapssl_install_routines( ld ) != 0 ) {
    ldap_perror( ld, "ldapssl_install_routines" );
    return( 1 );
}
/* Set up option in LDAP struct for using SSL */
if ( ldap_set_option( ld, LDAP_OPT_SSL, LDAP_OPT_ON ) != 0 ) {
    ldap_perror( ld, "ldap_set_option" );
    return( 1 );
}
/* Client can now perform LDAP operations on the secure LDAP server */
...
```

Handling Errors

The function `ldapssl_err2string()` provides support for special SSL error messages that are not handled by the normal error conversion routine `ldap_err2string()`. After calling any of the SSL initialization functions, you may convert SSL-specific errors codes to text strings by calling `ldapssl_err2string()`.

Installing Your Own SSL I/O Functions

The `ldapssl_init()` and `ldapssl_install_routines()` functions both set up the session to use the standard SSL I/O functions provided with the LDAP SDK for C. If you want to use your own SSL I/O functions, use the `ldap_io_fns` structure. The procedure to specify your own SSL I/O functions is:

1. Create an `ldap_io_fns` structure, and set the fields to point to your I/O functions.
2. Call `ldap_set_option()` to point to that structure.

Code Example 11-3 illustrates this.

Code Example 11-3 Setting Custom SSL I/O Functions

```
if (ldap_set_option( ld, LDAP_OPT_IO_FN_PTRS, &my_io_struct) != 0 ) {
    ldap_perror( ld, "ldap_set_option" );
    return( 1 );
}
```

Using Certificate-Based Client Authentication

Some LDAP servers may be configured to use certificate-based client authentication. In this case, the server requests that your client send a certificate to identify itself. This is the procedure to configure your client to use certificates for authentication:

1. Initialize your LDAP client by calling one of the following functions:
 - a. `ldapssl_clientauth_init()`
 - b. `ldapssl_advclientauth_init()`

You should call one of these functions to initialize your client instead of the `ldapssl_client_init()` function (Step a on page 221). Use `ldapssl_advclientauth_init()` if you need to specify the name and path of a security module database or if you need to specify the method used to verify the server's certificate.

NOTE	You can use these functions to initialize your client even if you do not plan to use certificate-based client authentication. They are equivalent to <code>ldapssl_client_init()</code> .

2. Initialize an LDAP session with the secure server by calling `ldapssl_init()`.
3. Enable your client to authenticate with the secure server by calling `ldapssl_enable_clientauth()`.
4. Perform a Simple Authentication and Security Layer (SASL) bind operation using the mechanism `EXTERNAL`. This indicates to the directory server that certificates should be used to authenticate clients.

NOTE	In all releases of Sun Java System Directory Server, if you perform a SASL bind operation and the server cannot find the corresponding directory entry for a client certificate, the server returns an <code>LDAP_INVALID_CREDENTIALS</code> result code with the error message <code>Client Certificate Mapping Failed</code> .

Using SASL Authentication

SASL provides the means to use mechanisms other than simple authentication and SSL to authenticate to Sun™ Java System LDAP SDK for C. This chapter describes the process of using a Simple Authentication and Security Layer (SASL) mechanism to authenticate an LDAP client to an LDAP server. It includes the following sections:

- Understanding SASL
- Determining Supported SASL Mechanisms
- Authenticating Using SASL

Understanding SASL

The ability to authenticate to an LDAP server using a SASL mechanism is a feature new to LDAPv3; LDAPv2 servers do not support this method of authentication. SASL is described in *RFC 2222 - Simple Authentication and Security Layer (SASL)* (<http://www.faqs.org/rfcs/rfc2222.html>).

Determining Supported SASL Mechanisms

To determine the SASL mechanisms supported by an LDAPv3 server, get the root DSE of the server, and check the `supportedSASLMechanisms` attribute. The values of this attribute are the names of the SASL mechanisms supported by the server.

NOTE If the root DSE does not have a `supportedSASLMechanisms` attribute, the server does not support any SASL mechanisms.

For information on getting the root DSE, see “Getting the Root DSE” on page 212 in Chapter 10, “Retrieving Server Information.”

Authenticating Using SASL

To authenticate to the server using a SASL mechanism, call one of the following:

- The synchronous `ldap_sasl_bind_s()` function.
- The asynchronous `ldap_sasl_bind()` function.

If you call the asynchronous function `ldap_sasl_bind()`, you need to call the `ldap_result()` and `ldap_parse_sasl_bind_result()` functions to get the result of the SASL bind operation.

Authentication using a SASL mechanism may take one or more round trips between your client and the server. (The server may send a number of *challenges* to the client.) You may need to call `ldap_sasl_bind_s()` several times (or `ldap_sasl_bind()`, `ldap_result()`, and `ldap_parse_sasl_bind_result()` several times) in order to respond to each server challenge.

NOTE	Before calling the function to perform a SASL bind operation, make sure to specify that your client is LDAPv3 compliant. If you do not, an <code>LDAP_NOT_SUPPORTED</code> result code is returned. See “Specifying the LDAP Version of Your Client” on page 65 in Chapter 3, “Writing an LDAP Client.”

Synchronous SASL Bind Operation

If you want to wait for the results of the SASL bind operation to complete before continuing, call the synchronous `ldap_sasl_bind_s()` function. This function sends a SASL bind request to the server and blocks other work until the server sends the results of the operation back to your client. `ldap_sasl_bind_s()` returns one of the following values:

- `LDAP_SUCCESS` if your client has successfully authenticated.
- `LDAP_SASL_BIND_IN_PROGRESS` if the server sends a challenge to your client. If you receive this result code, check the `servercredp` argument for the `berval` structure containing the server’s challenge. Call the `ldap_sasl_bind_s()` function again to send a response to that challenge.

- An LDAP error code, if a problem occurred or if authentication failed. See the documentation on the `ldap_sasl_bind_s()` function for a list of the possible result codes.

Asynchronous SASL Bind Operation

If you want to perform other work (in parallel) while waiting for the SASL bind operation to complete, do the following:

1. Call the asynchronous `ldap_sasl_bind()` function to send an LDAP SASL bind request.

This function returns an `LDAP_SUCCESS` result code if the request was successfully sent (or an LDAP result code if an error occurred while sending the request). The function also sets the `msgidp` argument to point to a message ID identifying the SASL bind operation.

2. Call the `ldap_result()` function and pass in this message ID to determine whether the server sent a response for this operation to your client.

The `ldap_result()` function uses the message ID to determine if the server sent a SASL bind response. The function passes back the response in an `LDAPMessage` structure.

3. Call the `ldap_parse_sasl_bind_result()` function to parse the `LDAPMessage` structure and retrieve information from the server's response.

If the server sent a challenge to your client, the challenge is specified in the `berval` structure passed back as the `servercredp` argument.

4. Call the `ldap_get_lderrno()` function to get the LDAP result code for the operation. The function can return one of the following values:

- a. `LDAP_SUCCESS` if your client successfully authenticated to the server.
- b. `LDAP_SASL_BIND_IN_PROGRESS` if the server sent a challenge to your client.

If the server returned an `LDAP_SASL_BIND_IN_PROGRESS` result code, check the `servercredp` argument for the `berval` structure containing the server's challenge.

- c. An LDAP error code, if a problem occurred or if authentication failed.

See the documentation on the return values of the `ldap_sasl_bind_s()` function for a list of result codes that the server can return for this operation.

5. If the result code is `LDAP_SASL_BIND_IN_PROGRESS` and the server passed back another challenge, determine the response to that challenge and call the `ldap_sasl_bind()` function again to send that response to the server.

You can call `ldap_result()` and `ldap_parse_sasl_bind_result()` again to get the next challenge sent from the server, if the result is `LDAP_SASL_BIND_IN_PROGRESS` again.

Code Example 12-1 is an LDAP client that authenticates using the SASL mechanism named `babsmechanism`.

Code Example 12-1 Authenticating over SASL

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "ldap.h"

int
main( int argc, char **argv )
{
    LDAP      *ld;
    LDAPMod    mod0;
    LDAPMod    mod1;
    LDAPMod    *mods[ 3 ];
    char        *vals0[ 2 ];
    char        *vals1[ 2 ];
    time_t      now;
    char        buf[ 128 ];
    struct berval cred;
    struct berval *servcred;
    int         version;
    /* get a handle to an LDAP connection */
    if ( (ld = ldap_init( "localhost", 389 )) == NULL ) {
        perror( "ldap_init" );
        return( 1 );
    }
    /* Set the LDAP protocol version supported by the client
       to 3. (By default, this is set to 2. SASL authentication
       is part of version 3 of the LDAP protocol.) */
    version = LDAP_VERSION3;
    ldap_set_option( ld, LDAP_OPT_PROTOCOL_VERSION, &version );
    /* authenticate */
    cred.bv_val = "magic";
    cred.bv_len = sizeof( "magic" ) - 1;
    if ( ldap_sasl_bind_s( ld, "uid=bjensen,ou=people,dc=example,dc=com", \
        "babsmechanism", &cred, NULL, NULL, &servcred ) != LDAP_SUCCESS ) {
        ldap_perror( ld, "ldap_sasl_bind_s" );
        return( 1 );
    }
    /* get and print the credentials returned by the server */
    printf( "Server credentials: %s\n", servcred->bv_val );
    /* construct the list of modifications to make */
}
```

Code Example 12-1 Authenticating over SASL

```

mod0.mod_op = LDAP_MOD_REPLACE;
mod0.mod_type = "mail";
vals0[0] = "babs@example.com";
vals0[1] = NULL;
mod0.mod_values = vals0;
mod1.mod_op = LDAP_MOD_ADD;
mod1.mod_type = "description";
time( &now );
sprintf( buf, "This entry was modified with the modattrs program on %s",
         ctime( &now ) );
/* Get rid of \n which ctime put on the end of the time string */
if ( buf[ strlen( buf ) - 1 ] == '\n' ) {
buf[ strlen( buf ) - 1 ] = '\0';
}
vals1[ 0 ] = buf;
vals1[ 1 ] = NULL;
mod1.mod_values = vals1;
mods[ 0 ] = &mod0;
mods[ 1 ] = &mod1;
mods[ 2 ] = NULL;
/* make the change */
if ( ldap_modify_s( ld, "uid=bjensen,ou=people,dc=example,dc=com", mods
)
    != LDAP_SUCCESS ) {
ldap_perror( ld, "ldap_modify_s" );
return( 1 );
}
ldap_unbind( ld );
printf( "modification was successful\n" );
return( 0 );
}

```


Working with LDAP Controls

This chapter explains how Lightweight Directory Access Protocol (LDAP) controls work and how to use those controls that are supported by Sun™ Java System LDAP SDK for C. It includes the following sections:

- How LDAP Controls Work
- Using Controls in the LDAP API
- Determining Supported Controls
- Using the Server-Side Sorting Control
- Using the Persistent Search Control
- Using the Entry Change Notification Control
- Using the Virtual List View Control
- Using the Manage DSA IT Control
- Using Password Policy Controls
- Using the Proxied Authorization Control

How LDAP Controls Work

The LDAPv3, as documented in *RFC 2251 - Lightweight Directory Access Protocol (v3)* (<http://www.faqs.org/rfcs/rfc2251.html>), allows clients and servers to use controls as a mechanism for extending an LDAP operation. A *control* is a way to specify additional information as part of a request and a response. For example, a client can send a control to a server as part of a search request to indicate that the server should sort the search results before sending the results back to the client.

NOTE Servers can also send controls back to clients. For example, Sun Java System Directory Server can send a control back to a client during the authentication process if the client's password has expired or is going to expire.

A control specifies the following information:

- A unique object identifier (OID) as defined by the creator of the control. The OID identifies the control. Table 13-1 on page 234 lists some common OIDs.
- An indication of whether or not the control is critical to the operation.
- Optional data related to the control. For example, the server-side sorting control needs the attributes that would be used for sorting search results.

When your client includes a control in a request for an LDAP operation, the server may respond in one of the following ways:

- If the server supports the control and if the control is appropriate to the operation, the server should make use of the control when performing the operation.
- If the server does not support the control type or if the control is not appropriate, the server should do one of the following:
 - If the control is marked as critical to the operation, the server should not perform the operation and should instead return the result code `LDAP_UNAVAILABLE_CRITICAL_EXTENSION`.
 - If the control is marked as not critical to the operation, the server should ignore the control and perform the operation.

NOTE If you plan to use a control, you need to make sure that the server supports the control. See "Determining Supported Controls" on page 234 for details.

Using Controls in the LDAP API

The LDAP SDK for C supports two types of controls:

- Server controls can be included in requests sent by clients and in responses sent by servers.
- Client controls affect the behavior of the SDK only and are never sent to the server.

In the SDK, a control is represented by an `LDAPControl` structure.

The fields in this structure represent the data in a control:

- `ldctl_oid` specifies the OID of the control.
- `ldctl_value` contains a `berval` structure containing data associated with the control.
- `ldctl_iscritical` specifies whether or not the control is critical to the operation. `LDAP_OPT_ON` indicates that the control is critical, and `LDAP_OPT_OFF` indicates that the control is not critical.

Code Example 13-1 is the `LDAPControl` structure definition. For more information on the `LDAPControl` structure, see Chapter 16, “Data Type Reference.”

Code Example 13-1 Example of `LDAPControl` Structure

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

You can either allocate and create the control yourself, or you can call a function to create the control. For example, you can call the `ldap_create_sort_control()` function to create a server-sorting control. To include a control in a request, call one of the LDAPv3 API functions (functions with names ending with `_ext` and `_ext_s`). These functions allow you to pass in an array of server controls and an array of client controls.

NOTE You can also include controls in a request by specifying the array of controls in the `LDAP_OPT_SERVER_CONTROLS` option. However, these controls will be sent to the server with every request. If the control is specific to a certain type of operation, you should use the `_ext` and `_ext_s` functions instead.

To retrieve any controls included in a server’s response, call the `ldap_parse_result()` function. You can then retrieve data from the returned controls yourself (by checking the fields of the `LDAPControl` structure) or by calling additional functions (such as `ldap_parse_sort_control()`).

When you are done working with a control or with an array of controls, you should free them from memory by calling the `ldap_control_free()` function or the `ldap_controls_free()` function.

Determining Supported Controls

According to the LDAPv3, servers should list any controls that they support in the `supportedControl` attribute in the root DSE.

NOTE See Chapter 10, “Retrieving Server Information” for information on the DSE.

Table 13-1 lists some of the OIDs for server controls that might be referenced in the `supportedControl` attribute.

Table 13-1 LDAPv3 Server Controls

OID of Control	Defined Name (in <code>ldap.h</code>)	Description of Control
2.16.840.1.113730.3.4.2	LDAP_CONTROL_MANAGEDSAIT	“Using the Manage DSA IT Control” on page 245
2.16.840.1.113730.3.4.3	LDAP_CONTROL_PERSISTENTSEARCH	“Using the Persistent Search Control” on page 242
2.16.840.1.113730.3.4.4	LDAP_CONTROL_PWEXPIRED	“Using Password Policy Controls” on page 245.
2.16.840.1.113730.3.4.5	LDAP_CONTROL_PWEXPIRING	“Using Password Policy Controls” on page 245
2.16.840.1.113730.3.4.9	LDAP_CONTROL_VLVREQUEST	“Using the Virtual List View Control” on page 245
1.2.840.113556.1.4.473	LDAP_CONTROL_SORTREQUEST	“Using the Server-Side Sorting Control” on page 237
2.16.840.1.113730.3.4.12	LDAP_CONTROL_PROXYAUTH	“Using the Proxied Authorization Control” on page 246

Code Example 13-2 is a simple command-line program that searches for the root DSE and prints the values of the `supportedControl` attribute.

Code Example 13-2 Sample Program to Search for Supported Controls

```

#include "ldap.h"
static char *usage = "Usage: listctrl -h <hostname> -p <portnumber>\n";

/* Associate OIDs of known controls with descriptions. */
struct oid2desc {
    char *oid;
    char *desc;
};
static struct oid2desc oidmap[] = {
    {LDAP_CONTROL_MANAGEDSAIT, "Manage DSA IT control"},
    {LDAP_CONTROL_SORTREQUEST, "Server-side sorting control"},
    {LDAP_CONTROL_PERSISTENTSEARCH, "Persistent search control"},
    {LDAP_CONTROL_VLVREQUEST, "Virtual list view control"},
    {LDAP_CONTROL_PWEXPIRED, "Password expired control"},
    {LDAP_CONTROL_PWEXPIRING, "Password expiration warning control"},
    { NULL, NULL }
};
int
main( int argc, char **argv )
{
    LDAP *ld;
    LDAPMessage *result, *e;
    char *hostname = NULL;
    char **vals;
    char *attrs[2];
    int i, j, c, portnumber = LDAP_PORT, rc;
    LDAPControl **serverctrls = NULL, **clntctrls = NULL;
    /* Parse the command line arguments. */
    while ( ( c = getopt( argc, argv, "h:p:" ) ) != -1 ) {
        switch ( c ) {
            case 'h':
                hostname = strdup( optarg );
                break;
            case 'p':
                portnumber = atoi( optarg );
                break;
            default:
                printf( "Unsupported option: %c\n", c );
                printf( usage );
                exit( 1 );
        }
    }
    /* By default, connect to localhost at port 389. */
    if ( hostname == NULL || hostname[0] == NULL ) {
        hostname = "localhost";
    }
    /* Initialize the connection. */
    if ( (ld = ldap_init( hostname, portnumber )) == NULL ) {
        perror( "ldap_init" );
        return( 1 );
    }
    /* Set automatic referral processing off. */
    if ( ldap_set_option( ld, LDAP_OPT_REFERRALS, LDAP_OPT_OFF )
        != LDAP_SUCCESS ) {

```

Code Example 13-2 Sample Program to Search for Supported Controls (*Continued*)

```

        ldap_perror( ld, "ldap_set_option" );
        return( 1 );
    }
    /* Search for the root DSE and retrieve only the
       supportedControl attribute. */
    attrs[ 0 ] = "supportedControl";
    attrs[ 1 ] = NULL;
    rc = ldap_search_ext_s( ld, "", LDAP_SCOPE_BASE, "(objectclass=*)",
        attrs, 0, serverctrls, clntctrls, NULL, NULL, &result );
    /* Check the search results. */
    switch( rc ) {
        /* If successful, the root DSE was found. */
        case LDAP_SUCCESS:
            break;
        /* If the root DSE was not found, the server does not comply
           with the LDAPv3 protocol. */
        case LDAP_PARTIAL_RESULTS:
        case LDAP_NO_SUCH_OBJECT:
        case LDAP_OPERATIONS_ERROR:
        case LDAP_PROTOCOL_ERROR:
            printf( "LDAP server %s:%d returned result code %d (%s).\n"
                "This server does not support the LDAPv3 protocol.\n",
                hostname, portnumber, rc, ldap_err2string( rc ) );
            return( 1 );
            break;
        /* If any other value is returned, an error must have occurred. */
        default:
            ldap_perror( ld, "ldap_search_ext_s" );
            return( 1 );
            break;
    }
    /* Get the root DSE from the results.
       Since there is only one root DSE, there
       should be only one entry in the results. */
    e = ldap_first_entry( ld, result );
    /* Get and print the values of the supportedControl attribute. */
    if ( e != NULL &&
        (vals = ldap_get_values( ld, e, "supportedControl" )) != NULL ) {
        printf( "\nControls Supported by %s:%d\n", hostname, portnumber );
        printf( "=====\n" );
        for ( i = 0; vals[i] != NULL; i++ ) {
            printf( "%s\n", vals[i] );
            for ( j = 0; oidmap[j].oid != NULL; j++ ) {
                if ( !strcmp( vals[i], oidmap[j].oid ) ) {
                    printf( "\t%s\n", oidmap[j].desc );
                }
            }
        }
        /* Free the values allocated by ldap_get_values(). */
        ldap_value_free( vals );
        printf( "\n" );
    }
    /* Free memory allocated by ldap_search_ext_s(). */
    ldap_msgfree( result );
    ldap_unbind( ld );

```

Code Example 13-2 Sample Program to Search for Supported Controls (*Continued*)

```
    return( 0 );
}
```

Using the Server-Side Sorting Control

The control with the OID 1.2.840.113556.1.4.473 (or LDAP_CONTROL_SORTREQUEST, as defined in the `ldap.h` header file) is a server-side sorting control. When you send a search request with this control to the server, the server should sort the results before sending them back to you.

NOTE The server-side sorting control is described in the Internet-Drafts file, *RFC 2891 - LDAP Control Extension for Server Side Sorting of Search Results* located at <http://www.faqs.org/rfcs/rfc2891.html>

Specifying the Sort Order

To specify the sort order of the results, call the `ldap_create_sort_keylist()` function to create a sort key list from a string in the following format:

```
[ - ]attrname[ :matching_rule_oid] ...
```

Where:

- *attrname* is the name of the attribute that you want to sort by. You can specify a space-delimited list of attribute names.
- *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 should be sorted in reverse order for that attribute. For example, the following string specifies that results should be sorted by last name (sn) first in ascending order. If multiple entries have the same last name, these entries are sorted by first name (givenname) in descending order:

```
"sn -givenname"
```

Passing this string to `ldap_create_sort_keylist()` creates a sort key list, which is an array of `LDAPsortkey` structures. You can use this to create the server-side sorting control.

Creating the Control

Next, to create the server-side sorting control, you pass the sort key list (the array of `LDAPsortkey` structures) to the `ldap_create_sort_control()` function. The function passes back a newly created sort control, an `LDAPControl` structure, which you can include in a search request.

NOTE You can specify whether or not the control is critical to the search operation. If the control is marked as critical and the server cannot sort the results, the server should not send back any entries. See “Interpreting the Results” on page 239 for more information on the ramifications of marking the control as critical.

After you call the `ldap_create_sort_control()` function and create the control, you should free the array of `LDAPsortkey` structures by calling `ldap_free_sort_keylist()`. When you are done receiving sorted results from the server, you should free the `LDAPControl` structure by calling `ldap_control_free()`.

Performing the Search

To specify that you want the server to sort the results, add the newly created server-sorting control to a NULL-terminated array of `LDAPControl` structures and pass this array to the `ldap_search_ext()` function or the `ldap_search_ext_s()` function. The server returns a result for the search operation and a response control. The *response control* indicates the success or failure of the sorting. To determine if sorting was successful:

1. Call `ldap_parse_result()` to parse the result of the search operation and retrieve any response controls sent back from the server.

Response controls are passed back in a NULL-terminated array of `LDAPControl` structures.

2. Pass this array of structures as an argument to `ldap_parse_sort_control()` to retrieve the LDAP result code for the sorting operation.

If the sorting operation fails, the server may also return the name of the attribute that caused the failure. The `ldap_parse_sort_control()` function also retrieves this name, if available.

3. Free the array by calling the `ldap_controls_free()` function when you are done parsing the array of response controls.

The server can return the result codes detailed in Table 13-2.

Table 13-2 Server-side Sorting Result Codes

Result Code	Description
LDAP_SUCCESS	The results were sorted successfully.
LDAP_OPERATIONS_ERROR	An internal server error occurred.
LDAP_TIMELIMIT_EXCEEDED	The maximum time allowed for a search was exceeded before the server finished sorting the results.
LDAP_STRONG_AUTH_REQUIRED	The server refused to send back the sorted search results because it requires you to use a stronger authentication method.
LDAP_ADMINLIMIT_EXCEEDED	There are too many entries for the server to sort.
LDAP_NO_SUCH_ATTRIBUTE	The sort key list specifies an attribute that does not exist.
LDAP_INAPPROPRIATE_MATCHING	The sort key list specifies a matching rule that is not recognized or appropriate.
LDAP_INSUFFICIENT_ACCESS	The server did not send the sorted results because the client has insufficient access rights.
LDAP_BUSY	The server is too busy to sort the results.
LDAP_UNWILLING_TO_PERFORM	The server is unable to sort the results.
LDAP_OTHER	This general result code indicates that the server failed to sort the results for a reason other than the ones listed above.

Interpreting the Results

Table 13-3 lists the kinds of results to expect from the LDAP server under different situations.

Table 13-3 Server Responses to Sorting Controls

Does the Server Support the Sort Control?	Is the Sort Control Marked As Critical?	Other Conditions	Results from LDAP Server
No	Yes	N/A	The server does not send back any entries.
	No		The server ignores the sorting control and returns the entries unsorted.

Table 13-3 Server Responses to Sorting Controls *(Continued)*

Does the Server Support the Sort Control?	Is the Sort Control Marked As Critical?	Other Conditions	Results from LDAP Server
Yes	Yes	The server cannot sort the results using the specified sort key list.	The server does not send back any entries. It sends back the sorting response control, which specifies the result code of the sort attempt and (optionally) the attribute type that caused the error.
	No		The server returns the entries unsorted. It sends back the sorting response control, which specifies the result code of the sort attempt and (optionally) the attribute type that caused the error.
	N/A (may or may not be marked as critical)	The server successfully sorted the entries.	The server sends back the sorted entries. It sends back the sorting response control, which specifies the result code of the sort attempt (LDAP_SUCCESS).
		The search itself failed (for any reason).	The server sends back a result code for the search operation. It does not send back the sorting response control.

Server-Sorting Control Sample Program

Code Example 13-3 uses the server-sorting control to get a list of all users in the directory, sorted in ascending order by last name, then in descending order by first name.

Code Example 13-3 Sample Program to Apply Server-sorting Control

```
#include <stdio.h>
#include "ldap.h"
/* Change these as needed. */
#define HOSTNAME "localhost"
#define PORTNUMBER 3890
int
main( int argc, char **argv )
{
    LDAP      *ld;
    LDAPMessage *result, *e;
    char      *attrfail, *matched = NULL, *errmsg = NULL;
```


Code Example 13-3 Sample Program to Apply Server-sorting Control (*Continued*)

```

char          **vals, **referrals;
int           rc, parse_rc, version;
unsigned long  rcode;
LDAPControl    *sortctrl = NULL;
LDAPControl    *requestctrls[ 2 ];
LDAPControl    **resultctrls = NULL;
LDAPsortkey    **sortkeylist;
/* Get a handle to an LDAP connection */
if ( (ld = ldap_init( HOSTNAME, PORTNUMBER ) ) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
}
version = LDAP_VERSION3;
ldap_set_option( ld, LDAP_OPT_PROTOCOL_VERSION, &version );
/* Create a sort key list that specifies the sort order of the results.
   Sort the results by last name first, then by first name. */
ldap_create_sort_keylist( &sortkeylist, "sn -givenname" );
/* Create the sort control. */
rc = ldap_create_sort_control( ld, sortkeylist, 1, &sortctrl );
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_create_sort_control: %s\n", ldap_err2string( rc )
);
    ldap_unbind( ld );
    return( 1 );
}
requestctrls[ 0 ] = sortctrl;
requestctrls[ 1 ] = NULL;
/* Search for all entries in Sunnyvale */
rc = ldap_search_ext_s( ld, "dc=example,dc=com", LDAP_SCOPE_SUBTREE,
    "(mail=*example.com*)", NULL, 0, requestctrls, NULL, NULL, 0, &result );
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_search_ext_s: %s\n", ldap_err2string( rc ) );
    ldap_unbind( ld );
    return( 1 );
}
parse_rc = ldap_parse_result( ld, result, &rc, &matched, &errmsg,
    &referrals, &resultctrls, 0 );
if ( parse_rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_parse_result: %s\n", ldap_err2string( parse_rc )
);
    ldap_unbind( ld );
    return( 1 );
}
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_search_ext_s: %s\n", ldap_err2string( rc ) );
    if ( errmsg != NULL && *errmsg != '\0' ) {
        fprintf( stderr, "%s\n", errmsg );
    }
    ldap_unbind( ld );
    return( 1 );
}
parse_rc = ldap_parse_sort_control( ld, resultctrls, &rcode, &attrfail );
if ( parse_rc != LDAP_SUCCESS ) {

```

Code Example 13-3 Sample Program to Apply Server-sorting Control (*Continued*)

```

    fprintf( stderr, "ldap_parse_sort_control: %s\n", ldap_err2string(
parse_rc ) );
    ldap_unbind( ld );
    return( 1 );
}

if ( rcode != LDAP_SUCCESS ) {
    fprintf( stderr, "Sort error: %s\n", ldap_err2string( rcode ) );
    if ( attrfail != NULL && *attrfail != '\0' ) {
        fprintf( stderr, "Bad attribute: %s\n", attrfail );
    }
    ldap_unbind( ld );
    return( 1 );
}
/* for each entry print out name + all attrs and values */
for ( e = ldap_first_entry( ld, result ); e != NULL;
      e = ldap_next_entry( ld, e ) ) {
    if ((vals = ldap_get_values( ld, e, "sn")) != NULL ) {
        if ( vals[0] != NULL ) {
            printf( "%s", vals[0] );
        }
        ldap_value_free( vals );
    }
    if ((vals = ldap_get_values( ld, e, "givenname")) != NULL ) {
        if ( vals[0] != NULL ) {
            printf( "\t%s", vals[0] );
        }
        ldap_value_free( vals );
    }
    printf( "\n" );
}
ldap_msgfree( result );
ldap_free_sort_keylist( sortkeylist );
ldap_control_free( sortctrl );
ldap_controls_free( resultctrls );
ldap_unbind( ld );
return( 0 );
}

```

Using the Persistent Search Control

The control with the OID 2.16.840.1.113730.3.4.3 (or LDAP_CONTROL_PERSISTENTSEARCH as defined in the ldap.h header file) is the persistent search control. A *persistent search* is an ongoing search operation that allows your LDAP client to get notification of changes to the directory.

NOTE

The persistent search control is described in the Internet-Drafts file, *Persistent Search: A Simple LDAP Change Notification Mechanism* located at <http://www.ietf.org/proceedings/01mar/I-D/ldapext-psearch-03.txt>.

To use persistent searching for change notification, you create a *persistent search* control that specifies the types of changes that you want to track. You include the control in a search request. If an entry in the directory is changed, the server determines if the entry matches the search criteria in your request and if the change is the type of change that you are tracking. If both of these are true, the server sends the entry to your client.

To create a persistent search control, call `ldap_create_persistentsearch_control()` as illustrated in Code Example 13-4.

Code Example 13-4 Calling `ldap_create_persistentsearch_control()`

```
int ldap_create_persistentsearch_control( LDAP *ld,
    int changetypes, int changesonly, int return_echg_ctls,
    char ctl_iscritical, LDAPControl **ctrlp );
```

You can specify the following information:

- `changetypes` specifies the type of change you want to track. You can specify any of the following (or any combination of the following using a bitwise OR operator):
 - `LDAP_CHANGETYPE_ADD` indicates that you want to track added entries.
 - `LDAP_CHANGETYPE_DELETE` indicates that you want to track deleted entries.
 - `LDAP_CHANGETYPE_MODIFY` indicates that you want to track modified entries.
 - `LDAP_CHANGETYPE_MODDN` indicates that you want to track renamed entries.
 - `LDAP_CHANGETYPE_ANY` indicates that you want to track all changes to entries.
- `changesonly` indicates whether or not you want the server to return all entries that initially matched the search criteria (0 to return all entries, or non-zero to return only the entries that change).

- `return_echg_ctrls` indicates whether or not you want entry change notification controls included with every modified entry returned by the server (non-zero to return entry change notification controls).

NOTE You can use this control in conjunction with an "entry change notification" control. See "Using the Entry Change Notification Control" on page 244 for details.

The function passes back an `LDAPControl` structure representing the control in the `ctrlp` parameter. You can add the newly created control to a NULL-terminated array of `LDAPControl` structures and pass this array to the `ldap_search_ext()` function.

To end the persistent search, you can either call the `ldap_abandon_ext()` function to abandon the search operation, or you can call the `ldap_unbind()` function to disconnect from the server.

NOTE For an example detailing how to perform a persistent search, refer to the example provided with the LDAP SDK for C in *DSRK_base/lib/ldapsdk/examples* called `psearch.c`.

Using the Entry Change Notification Control

The control with the OID 2.16.840.1.113730.3.4.7 (or `LDAP_CONTROL_ENTRYCHANGE`, as defined in the `ldap.h` header file) is the "entry change notification" control. This control contains additional information about the change made to the entry, including the type of change made, the change number (which corresponds to an item in the server's change log, if the server supports a change log), and, if the entry was renamed, the old DN of the entry.

You use this control in conjunction with a persistent search control. If you have specified the preference for returning entry change notification controls, the server includes an entry change notification control with each entry found by the search. To retrieve and parse an entry change notification control included with an entry:

1. Pass the `LDAPMessage` structure that represents an entry to the `ldap_get_entry_controls()` function.
2. Pass the entry change notification control to the `ldap_parse_entrychange_control()` function.

For more information, see "Using the Persistent Search Control" on page 242.

Using the Virtual List View Control

The control with the OID 2.16.840.1.113730.3.4.9 (or `LDAP_CONTROL_VLVREQUEST`, as defined in the `ldap.h` header file) is a virtual list view control. When you send a search request with this control and a server-side sorting control to the server, the server should sort the results and return the specified subset of entries back to your client.

NOTE The virtual list view control is described in the Internet-Drafts file, *LDAP Extensions for Scrolling View Browsing of Search Results* located at <http://www.ietf.org/proceedings/01mar/I-D/ldapext-ldapv3-vlv-04.txt>.

Using the Manage DSA IT Control

The control with the OID 2.16.840.1.113730.3.4.2 (or `LDAP_CONTROL_MANAGEDSAIT`, as defined in the `ldap.h` header file) is the manage DSA IT control. You can use this control to manage search references in the directory. To create this control, create an `LDAPControl` structure and set the `ldctl_oid` field to 2.16.840.1.113730.3.4.2.

When you add this control to the array of `LDAPControl` structures that you pass to a function (for example, `ldap_search_ext()` or `ldap_modify_ext()`), the server treats search references as ordinary entries. Rather than returning a reference to you, the server returns the entry containing the reference. This allows your client application to manage search references in the directory.

NOTE The manage DSA IT control is described in the Internet-Drafts file, *RFC 2891 - LDAP Control Extension for Server Side Sorting of Search Results* located at <http://www.faqs.org/rfcs/rfc2891.html>

Using Password Policy Controls

Sun Java System Directory Server uses two server response controls to send information back to a client after an LDAP bind operation:

1. The control with the OID 2.16.840.1.113730.3.4.4 (or `LDAP_CONTROL_PWEXPIRED`, as defined in the `ldap.h` header file) is the expired password control. It is used if the server is configured to require users to change their passwords when first logging in and whenever their password

has been reset. If the user is logging in for the first time or their password has been reset, the server sends this control to indicate that the client needs to change the password immediately. At this point, the only operation that the client can perform is to change the user's password. If the client requests any other operation, the server sends back an `LDAP_UNWILLING_TO_PERFORM` result code with an expired password control.

2. The control with the OID `2.16.840.1.113730.3.4.5` (or `LDAP_CONTROL_PWEXPIRING`, as defined in the `ldap.h` header file) is the password expiration warning control. This control is used if the server is configured to expire user passwords after a certain amount of time. The server sends this control back to the client if the client binds using a password that will soon expire. The `ldctl_value` field of the `LDAPControl` structure specifies the number of seconds before the password will expire.

To get these server response controls when binding, you can:

- Call `ldap_simple_bind()` to send a request for an asynchronous bind operation.
- Call `ldap_result()` to get the results of the operation.
- Call `ldap_parse_result()` to parse the result and retrieve the server response controls from the result as an array of `LDAPControl` structures.

You can then check the `ldctl_oid` field to determine the OID of the control and the `ldctl_value` field for any data included in the control.

Using the Proxied Authorization Control

Proxied authorization is an extension to the LDAPv3 that allows a bound client to assume the identity of another directory entity without rebinding. This allows the client to perform operations as if it were bound as the proxied directory entity. All directory access (read, write, search, compare, delete, and add) is supported by proxied authorization. For example, suppose a client is bound as `uid=bjensen,ou=Engineering,dc=example,dc=com`. `bjensen` does not have the right to search the `ou=Marketing,dc=example,dc=com` tree. However, `uid=lboyd,ou=Marketing,dc=example,dc=com` does have rights to search the Marketing tree and, `lboyd` grants proxy rights to `bjensen`. In this case, `bjensen` may bind as herself, assume the identity of `lboyd`, and then search the Marketing tree.

This feature is intended as a performance and administrative benefit for certain types of directory usage. Specifically, applications that want to allow many clients to access or modify specific directory data without rebinding as another directory entity may want to use this feature.

CAUTION Be aware that other directory servers might not support this feature. Sun Java System Directory Server access control (including the proxy right) is fully described in the *Sun ONE Directory Server Administration Guide* located at <http://docs.sun.com/doc/816-6698-10>.

The Proxy Right

Proxied authorization adds an additional access right: *proxy*. If an entry grants the proxy right, then the entity to which that right is granted may assume the identity of the granting entity. For example, if you wanted to allow `uid=bjensen` the right to proxy as `uid=lboyd`, add the Proxy Right access control instruction (ACI) illustrated in Code Example 13-5 to your directory. This ACI allows `bjensen` to assume the identity of `lboyd` for all directory operations giving `bjensen` the permission to do to the directory whatever `lboyd` has permission to do.

Code Example 13-5 Proxy Right ACI

```
aci: (target = "ldap:///uid=lboyd,ou=Marketing,dc=example,dc=com")
(targetattr=*)
(version 3.0; aci "grant bjensen the right to proxy as lboyd";
allow(proxy)
userdn="ldap:///uid=bjensen,ou=Engineering,dc=example,dc=com";)
```

The Proxied Authorization Control

To support proxied authorization (an extension to the LDAPv3), the proxy authorization control has been added to the LDAP SDK for C in the form of the `ldap_create_proxyauth_control()` function. You use this function to create the control that allows a bound entity to assume the identity of another directory entry.

Proxy authorization is an optional LDAP server feature; it may not be supported on all LDAP servers. You should call the proxy authorization control function only when interacting with LDAP servers that support this LDAPv3 extension. You can check on the support of this control by looking at the rootDSE supportedControl attribute. For example, the following command uses the `ldapsearch` utility to display the rootDSE:

```
ldapsearch -v -h hostname -p 389 -b "" -s base ""
```

In order for the control to work, the LDAP server that you are connecting to must support the server control for proxied authorization (OID 2.16.840.1.113730.3.4.12, or `LDAP_CONTROL_PROXYAUTH` as defined in the `ldap.h` header file).

Proxy Authorization Sample Program

Code Example 13-6 creates an LDAP connection, sets the proxied authorization control, binds to the directory, and then performs a search operation using the proxied authorization control. A more complete example is also available with the SDK example files located in `DSRK_base/lib/ldapcsdk/examples`.

Code Example 13-6 Proxied Authorization Control Example

```
#include <ldap.h>

int          version;
LDAP         *ld;
LDAPControl  *requestctrls[ 2 ];
LDAPControl  *pactrl = NULL;

/* Customize the following host and bind information for your site. */
int          port=389;
char         *host="directory.example.com";
char         *baseDN="dc=example,dc=com";

/* Proxied auth specific information.
   proxyDN is the entity that will be proxied.
   bindDN and bindpw is for the bind entity that will use the proxyDN. */
char         *proxyDN = "uid=lboyd,ou=marketing,dc=example,dc=com";
char         *bindDN = "uid=bjensen,ou=engineering,dc=example,dc=com";
char         *bindpw = "password";

/* Do general LDAP init stuff */
/* Get a handle to an LDAP connection */
if ( (ld = ldap_init( host, port ) ) == NULL ) {
    printf("ldap_init did not return a conn handle.\n");
    return;
}

/* set version to ldap version 3 */
```


Code Example 13-6 Proxied Authorization Control Example (*Continued*)

```

version = LDAP_VERSION3;
ldap_set_option( ld, LDAP_OPT_PROTOCOL_VERSION, &version );

/* authenticate to the directory */
if ( ldap_simple_bind_s( ld, bindDN, bindpw ) != LDAP_SUCCESS ) {
    printf("ldap_simple_bind_s failed");
    return (-1);
}

/* create the proxied authorization control */
if ( ldap_create_proxyauth_control( ld, proxyDN, 1, &pactrl ) ) {
    printf("ldap_create_proxyauth_control failed.\n");
    if ( ldap_unbind( ld ) != LDAP_SUCCESS ) {
        printf("ldap_unbind failed\n");
    }
    return(-1);
}

requestctrls[ 0 ] = pactrl;
requestctrls[ 1 ] = NULL;

/* Perform the search using the control */
printf("Searching for %s with the proxy auth control.\n", proxyDN);
if ( ldap_search_ext_s( ld, proxyDN, LDAP_SCOPE_SUBTREE, "(objectclass=*)",
    NULL, 0, requestctrls, NULL, NULL, LDAP_NO_LIMIT, &results ) !=
    LDAP_SUCCESS ) {
    printf("ldap_search_ext failed.\n");
    printf("Something is wrong with proxied auth.\n");
} else {
    print_search_results(ld, results);
}

```


Writing Multithreaded Clients

The Sun™ Java System LDAP SDK for C contains the functionality for multi-threaded clients to access the same data types. This chapter explains how to write multi-threaded LDAP clients. It consists of the following sections:

- Specifying Thread Functions
- Pthreads Client Application

Specifying Thread Functions

If you are writing a multi-threaded client with different threads access the same LDAP structure, you need to set up the session so that the threads do not interfere with each other. For example, in a single-threaded LDAP client, the LDAP structure contains the error code of the last LDAP operation. In a multi-threaded client, you need to set up the session so that each thread can have its own error code.

The `LDAP_OPT_THREAD_FN_PTRS` session option lets you set up an `ldap_thread_fns` structure identifying the functions that are called in multi-threaded environments (for example, functions to lock and unlock critical sections of code and to get and set errors). Because this structure lets you specify these functions, you can use the LDAP SDK for C in different types of threading environments.

Setting Up the `ldap_thread_fns` Structure

If you are writing a multi-threaded client in which different threads use the same LDAP connection, you need to set up the `ldap_thread_fns` structure and identify the functions that you want to use. Code Example 14-1 illustrates the `ldap_thread_fns` structure.

Code Example 14-1 ldap_thread_fns Structure

```
struct ldap_thread_fns {
    LDAP_TF_MUTEX_ALLOC_CALLBACK    *ltf_mutex_alloc;
    LDAP_TF_MUTEX_FREE_CALLBACK     *ltf_mutex_free;
    LDAP_TF_MUTEX_LOCK_CALLBACK     *ltf_mutex_lock;
    LDAP_TF_MUTEX_UNLOCK_CALLBACK   *ltf_mutex_unlock;
    LDAP_TF_GET_ERRNO_CALLBACK       *ltf_get_errno;
    LDAP_TF_SET_ERRNO_CALLBACK       *ltf_set_errno;
    LDAP_TF_GET_LDERRNO_CALLBACK     *ltf_get_lderrno;
    LDAP_TF_SET_LDERRNO_CALLBACK     *ltf_set_lderrno;
    void                             *ltf_lderrno_arg;
};
```

The fields of ldap_thread_fns structure are described in Table 14-1.

Table 14-1 ldap_thread_fns Field Descriptions

Field	Description
*ltf_mutex_alloc	Function pointer for allocating a mutex. This function is called by the client when needed if the function pointer is not NULL.
*ltf_mutex_free	Function pointer for freeing a mutex. This function is called by the client when needed if the function pointer is not NULL.
*ltf_mutex_lock	Function pointer for locking critical sections of code. This function is called by the client when needed if the function pointer is not NULL.
*ltf_mutex_unlock	Function pointer for unlocking critical sections of code. This function is called by the client when needed if the function pointer is not NULL.
*ltf_get_errno	Function pointer for getting the value of the errno variable. This function is called by the client when needed if the function pointer is not NULL. In a threaded environment, errno is typically redefined so that it has a value for each thread, rather than a global value for the entire process. This redefinition is done at compile time. Because the libldap library does not know what method your code and threading environment will use to get the value of errno for each thread, it calls this function to return the value of errno.
*ltf_set_errno	Function pointer for setting the value of the errno variable. This function is called by the client when needed if the function pointer is not NULL. In a threaded environment, errno is typically redefined so that it has a value for each thread, rather than a global value for the entire process. This redefinition is done at compile time. Because the libldap library does not know what method your code and threading environment will use to get the value of errno for each thread, it calls this function to set the value of errno.

Table 14-1 ldap_thread_fns Field Descriptions (*Continued*)

Field	Description
*ltf_get_lderrno	Function pointer for getting error values from calls to functions in the libldap library. This function is called by the client when needed if the function pointer is not NULL. If this function pointer is not set, the libldap library records these errors in fields in the LDAP structure.
*ltf_set_lderrno	Function pointer for setting error values from calls to functions in the libldap library. This function is called by the client when needed if the function pointer is not NULL. If this function pointer is not set, the libldap library records these errors in fields in the LDAP structure.
*ltf_lderrno_arg	Additional parameter passed to the functions for getting and setting error values from calls to functions in the libldap library. *ltf_get_lderrno and *ltf_set_lderrno identify these functions.

Setting Up the ldap_extra_thread_fns Structure

The LDAP SDK for C provides a structure, `ldap_extra_thread_fns`, which specifies additional thread functions for locking and for *semaphores* (a protected variable). The `ldap_extra_thread_fns` structure is defined in Code Example 14-2.

Code Example 14-2 Declaring ldap_extra_thread_fns Structure

```
struct ldap_extra_thread_fns {
    LDAP_TF_MUTEX_TRYLOCK_CALLBACK *ltf_mutex_trylock;
    LDAP_TF_SEMA_ALLOC_CALLBACK *ltf_sema_alloc;
    LDAP_TF_SEMA_FREE_CALLBACK *ltf_sema_free;
    LDAP_TF_SEMA_WAIT_CALLBACK *ltf_sema_wait;
    LDAP_TF_SEMA_POST_CALLBACK *ltf_sema_post;
    LDAP_TF_THREADID_CALLBACK *ltf_threadid_fn;
};
```

The LDAP SDK for C supports only the `LDAP_TF_THREADID_CALLBACK` *ltf_threadid_fn function. (If any of the other extra thread callback functions are set, they will be ignored.) The supported function must return an identifier that is unique to the calling thread, like `pthread_self()` does. You use this function callback in a multi-threaded application to improve the performance of thread locking.

Setting Session Options

After you set up the `ldap_thread_fns` structure, you need to associate the structure with the current session. Call the `ldap_set_option()` function and pass `LDAP_OPT_THREAD_FN_PTRS` as the value of the option parameter. Pass a pointer to the `ldap_thread_fns` structure as the value of the `optdata` parameter. Code Example 14-3 sets up an `ldap_thread_fns` structure for an LDAP session.

Code Example 14-3 Setting Up an `ldap_thread_fns` Structure

```
#include <stdio.h>
#include <malloc.h>
#include <errno.h>
#include <pthread.h>
#include <ldap.h>

struct ldap_thread_fns tfns;
...
/* Set up the ldap_thread_fns structure with pointers
   to the functions that you want called */
memset( &tfns, '\0', sizeof(struct ldap_thread_fns) );

/* Specify the functions that you want called */

/* Call the my_mutex_alloc() function whenever mutexes
   need to be allocated */
tfns.ltf_mutex_alloc = (void (*)(void)) my_mutex_alloc;

/* Call the my_mutex_free() function whenever mutexes
   need to be destroyed */
tfns.ltf_mutex_free = (void (*)(void *)) my_mutex_free;

/* Call the pthread_mutex_lock() function whenever a
   thread needs to lock a mutex. */
tfns.ltf_mutex_lock = (int (*)(void *)) pthread_mutex_lock;

/* Call the pthread_mutex_unlock() function whenever a
   thread needs to unlock a mutex. */
tfns.ltf_mutex_unlock = (int (*)(void *)) pthread_mutex_unlock;

/* Call the get_errno() function to get the value of errno */
tfns.ltf_get_errno = get_errno;

/* Call the set_errno() function to set the value of errno */
tfns.ltf_set_errno = set_errno;

/* Call the get_ld_error() function to get error values from
   calls to functions in the libldap library */
tfns.ltf_get_lderrno = get_ld_error;

/* Call the set_ld_error() function to set error values for
   calls to functions in the libldap library */
tfns.ltf_set_lderrno = set_ld_error;
```

Code Example 14-3 Setting Up an `ldap_thread_fns` Structure (*Continued*)

```

/* Don't pass any extra parameter to the functions for
   getting and setting libldap function call errors */
tfns.ltf_lderrno_arg = NULL;
...
/* Set the session option that specifies the functions to call for
   multi-threaded clients */
if (ldap_set_option( ld, LDAP_OPT_THREAD_FN_PTRS, (void *) &tfns)!= 0) {
    ldap_perror( ld, "ldap_set_option: thread pointers" );
}
...

```

If you also set up the `ldap_extra_thread_fns` structure, you can associate it with the current session by calling `ldap_set_option()` function and passing `LDAP_OPT_EXTRA_THREAD_FN_PTRS` as the value of the option parameter. You must also pass a pointer to the `ldap_extra_thread_fns` structure as the value of the `optdata` parameter.

Pthreads Client Application

Code Example 14-4, which uses *pthread*s (POSIX threads) on the Solaris™ platform, is the source code for a multi-threaded client. The client connects to a specified LDAP server and creates several threads to perform multiple search and update operations simultaneously on the directory.

Code Example 14-4 Using POSIX Threading Under Solaris

```

#include <stdio.h>
#include <malloc.h>
#include <errno.h>
#include <pthread.h>
#include <ldap.h>
#include <synch.h>

/* Authentication and search information. */
#define NAME      "cn=Directory Manager"
#define PASSWORD  "rtfm11111"
#define BASE      "dc=example,dc=com"
#define SCOPE      LDAP_SCOPE_SUBTREE

/* Function declarations */
static void *search_thread();
static void *modify_thread();
static void *add_thread();
static void *delete_thread();

```

Code Example 14-4 Using POSIX Threading Under Solaris (*Continued*)

```

static void set_ld_error();
static int  get_ld_error();
static void set_errno();
static int  get_errno();
static void tsd_setup();

/* Linked list of LDAPMessage structs for search results. */
typedef struct ldapmsgwrapper {
    LDAPMessage      *lmw_messagep;
    struct ldapmsgwrapper *lmw_next;
} ldapmsgwrapper;

LDAP      *ld;
pthread_key_t  key;

main( int argc, char **argv )
{
    pthread_attr_t  attr;
    pthread_t  search_tid, search_tid2, search_tid3, search_tid4;
    pthread_t  modify_tid, add_tid, delete_tid;
    void      *status;
    struct ldap_thread_fns  tfns;
    struct ldap_extra_thread_fns  extrafns;
    int rc;

    /* Check command-line syntax. */
    if ( argc != 3 ) {
        fprintf( stderr, "usage: %s <host> <port>\n", argv[0] );
        exit( 1 );
    }

    /* Create a key. */
    if ( pthread_key_create( &key, free ) != 0 ) {
        perror( "pthread_key_create" );
    }
    tsd_setup();

    /* Initialize the LDAP session. */
    if ( (ld = ldap_init( argv[1], atoi( argv[2] ) )) == NULL ) {
        perror( "ldap_init" );
        exit( 1 );
    }

    /* Set the function pointers for dealing with mutexes
       and error information. */
    memset( &tfns, '\0', sizeof(struct ldap_thread_fns) );
    tfns.ltf_mutex_alloc = (void (*)(void)) my_mutex_alloc;
    tfns.ltf_mutex_free = (void (*)(void*)) my_mutex_free;
    tfns.ltf_mutex_lock = (int (*)(void*)) pthread_mutex_lock;
    tfns.ltf_mutex_unlock = (int (*)(void*)) pthread_mutex_unlock;
    tfns.ltf_get_errno = get_errno;
    tfns.ltf_set_errno = set_errno;
    tfns.ltf_get_lderrno = get_ld_error;
    tfns.ltf_set_lderrno = set_ld_error;
    tfns.ltf_lderrno_arg = NULL;

```


Code Example 14-4 Using POSIX Threading Under Solaris (*Continued*)

```

/* Set up this session to use those function pointers. */
rc = ldap_set_option( ld, LDAP_OPT_THREAD_FN_PTRS, (void *) &tfns );
if ( rc < 0 ) {
    fprintf( stderr, "ldap_set_option (LDAP_OPT_THREAD_FN_PTRS): %s\n",
ldap_err2string( rc ) );
    exit( 1 );
}

/* Set the function pointers for working with semaphores. */
memset( &extrafns, '\0', sizeof(struct ldap_extra_thread_fns) );
extrafns.ltf_mutex_trylock = (int (*)(void *)) = null;
extrafns.ltf_sema_alloc = (void *(*)(void)) = null;
extrafns.ltf_sema_free = (void (*)(void *)) = null;
extrafns.ltf_sema_wait = (int (*)(void *)) = null;
extrafns.ltf_sema_post = (int (*)(void *)) = null;
extrafns.ltf_threadid_fn = (void * (*)(void)) pthread_self;
/* Set up this session to use those function pointers. */
rc = ldap_set_option( ld, LDAP_OPT_EXTRA_THREAD_FN_PTRS, (void *)
&extrafns );
if ( rc < 0 ) {
    fprintf( stderr, "ldap_set_option (LDAP_OPT_EXTRA_THREAD_FN_PTRS):
%s\n", ldap_err2string( rc ) );
    exit( 1 );
}

/* Attempt to bind to the server. */
rc = ldap_simple_bind_s( ld, NAME, PASSWORD );
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "ldap_simple_bind_s: %s\n", ldap_err2string( rc ) );
    exit( 1 );
}

/* Initialize the attribute. */
if ( pthread_attr_init( &attr ) != 0 ) {
    perror( "pthread_attr_init" );
    exit( 1 );
}

/* Specify that the threads are joinable. */
pthread_attr_setdetachstate( &attr, PTHREAD_CREATE_JOINABLE );

/* Create seven threads: one for adding, one for modifying,
one for deleting, and four for searching. */
if ( pthread_create( &search_tid, &attr, search_thread, "1" ) != 0 ) {
    perror( "pthread_create search_thread" );
    exit( 1 );
}
if ( pthread_create( &modify_tid, &attr, modify_thread, "2" ) != 0 ) {
    perror( "pthread_create modify_thread" );
    exit( 1 );
}
if ( pthread_create( &search_tid2, &attr, search_thread, "3" ) != 0 ) {
    perror( "pthread_create search_thread2" );
}

```

Code Example 14-4 Using POSIX Threading Under Solaris (*Continued*)

```

    exit( 1 );
}
if ( pthread_create( &add_tid, &attr, add_thread, "4" ) != 0 ) {
    perror( "pthread_create add_thread" );
    exit( 1 );
}
if ( pthread_create( &search_tid3, &attr, search_thread, "5" ) != 0 ) {
    perror( "pthread_create search_thread3" );
    exit( 1 );
}
if ( pthread_create( &delete_tid, &attr, delete_thread, "6" ) != 0 ) {
    perror( "pthread_create delete_thread" );
    exit( 1 );
}
if ( pthread_create( &search_tid4, &attr, search_thread, "7" ) != 0 ) {
    perror( "pthread_create search_thread4" );
    exit( 1 );
}

/* Wait until these threads exit. */
pthread_join( modify_tid, &status );
pthread_join( add_tid, &status );
pthread_join( delete_tid, &status );
pthread_join( search_tid, &status );
pthread_join( search_tid2, &status );
pthread_join( search_tid3, &status );
pthread_join( search_tid4, &status );
}

/* Thread for searching the directory.
   The results are not printed out. */
static void *
search_thread( char *id )
{
    LDAPMessage *res;
    LDAPMessage *e;
    char *a;
    char **v;
    char *dn;
    BerElement *ber;
    int i, rc, parse_rc, msgid, finished;
    int num_entries, num_refs;
    void *tsd;
    struct timeval zerotime;
    zerotime.tv_sec = zerotime.tv_usec = 0L;

    printf( "Starting search_thread %s.\n", id );
    tsd_setup();
    /* Continually search the directory. */
    for ( ;; ) {
        printf( "Thread %s: Searching...\n", id );
        finished = 0;
        num_entries = 0;
        num_refs = 0;
        rc = ldap_search_ext( ld, BASE, SCOPE, "(objectclass=*)",

```

Code Example 14-4 Using POSIX Threading Under Solaris (*Continued*)

```

    NULL, 0, NULL, NULL, NULL, LDAP_NO_LIMIT, &msgid );
if ( rc != LDAP_SUCCESS ) {
    fprintf( stderr, "Thread %s error: ldap_search: %s\n",
        id, ldap_err2string( rc ) );
    continue;
}

/* Iterate through the results. In this example,
   we don't print out all the results. (It's easier
   to see the output from the other threads this way.) */
while ( !finished ) {
    rc = ldap_result( ld, msgid, LDAP_MSG_ONE, &zerotime, &res );
    switch ( rc ) {
        case -1:
            rc = ldap_get_lderrno( ld, NULL, NULL );
            fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
            finished = 1;
            break;
        case 0:
            break;
        /* Keep track of the number of entries found. */
        case LDAP_RES_SEARCH_ENTRY:
            num_entries++;
            break;
        /* Keep track of the number of search references. */
        case LDAP_RES_SEARCH_REFERENCE:
            num_refs++;
            break;
        case LDAP_RES_SEARCH_RESULT:
            finished = 1;
            parse_rc = ldap_parse_result( ld, res, &rc, NULL, NULL, NULL, NULL,
1 );
            if ( parse_rc != LDAP_SUCCESS ) {
                fprintf( stderr, "Thread %s error: can't parse result code.\n", id
);
                break;
            } else {
                if ( rc != LDAP_SUCCESS ) {
                    fprintf( stderr, "Thread %s error: ldap_search: %s\n", id,
ldap_err2string( rc ) );
                } else {
                    printf( "Thread %s: Got %d results and %d references.\n", id,
num_entries, num_refs );
                }
            }
            break;
        default:
            break;
    }
}
}

/* Thread for modifying directory entries.

```

Code Example 14-4 Using POSIX Threading Under Solaris (*Continued*)

```

This thread searches for entries and randomly selects entries from
the search results for modification. */
static void *
modify_thread( char *id )
{
    LDAPMessage      *res;
    LDAPMessage      *e;
    int               i, modentry, num_entries, msgid, rc, parse_rc, finished;
    LDAPMod           mod;
    LDAPMod           *mods[2];
    char              *vals[2];
    char              *dn;
    ldapmsgwrapper    *list, *lmwp, *lastlmwp;
    struct timeval     zerotime;
    zerotime.tv_sec = zerotime.tv_usec = 0L;

    printf( "Starting modify_thread %s.\n", id );
    tsd_setup();
    rc = ldap_search_ext( ld, BASE, SCOPE, "(objectclass=*)",
        NULL, 0, NULL, NULL, NULL, LDAP_NO_LIMIT, &msgid );
    if ( rc != LDAP_SUCCESS ) {
        fprintf( stderr, "Thread %s error: Modify thread: "
            "ldap_search_ext: %s\n", id, ldap_err2string( rc ) );
        exit( 1 );
    }
    list = lastlmwp = NULL;
    finished = 0;
    num_entries = 0;
    while ( !finished ) {
        rc = ldap_result( ld, msgid, LDAP_MSG_ONE, &zerotime, &res );
        switch ( rc ) {
            case -1:
                rc = ldap_get_lderrno( ld, NULL, NULL );
                fprintf( stderr, "ldap_result: %s\n", ldap_err2string( rc ) );
                exit( 1 );
                break;
            case 0:
                break;

            /* Keep track of the number of entries found. */
            case LDAP_RES_SEARCH_ENTRY:
                num_entries++;
                if ( ( lmwp = (ldapmsgwrapper *)
                    malloc( sizeof( ldapmsgwrapper ) ) ) == NULL ) {
                    fprintf( stderr, "Thread %s: Modify thread: Cannot malloc\n", id );
                    exit( 1 );
                }
                lmwp->lmw_messagep = res;
                lmwp->lmw_next = NULL;
                if ( lastlmwp == NULL ) {
                    list = lastlmwp = lmwp;
                } else {
                    lastlmwp->lmw_next = lmwp;
                }
                lastlmwp = lmwp;
        }
    }
}

```

Code Example 14-4 Using POSIX Threading Under Solaris (*Continued*)

```

        break;
    case LDAP_RES_SEARCH_REFERENCE:
        break;
    case LDAP_RES_SEARCH_RESULT:
        finished = 1;
        parse_rc = ldap_parse_result( ld, res, &rc, NULL, NULL, NULL, NULL, 1
);
        if ( parse_rc != LDAP_SUCCESS ) {
            fprintf( stderr, "Thread %s error: can't parse result code.\n", id );
            exit( 1 );
        } else {
            if ( rc != LDAP_SUCCESS ) {
                fprintf( stderr, "Thread %s error: ldap_search: %s\n", id,
ldap_err2string( rc ) );
            } else {
                printf( "Thread %s: Got %d results.\n", id, num_entries );
            }
        }
        break;
    default:
        break;
}
}

/* Set up the modifications to be made. */
mods[0] = &mod;
mods[1] = NULL;
vals[0] = "bar";
vals[1] = NULL;

/* Modify randomly selected entries. */
for ( ;; ) {

    /* Randomly select the entries. */
    modentry = rand() % num_entries;
    for ( i = 0, lmwp = list; lmwp != NULL && i < modentry;
        i++, lmwp = lmwp->lmw_next ) {
        /* Keep iterating. */
    }
    if ( lmwp == NULL ) {
        fprintf( stderr,
            "Thread %s: Modify thread could not find entry %d of %d\n",
            id, modentry, num_entries );
        continue;
    }
    e = lmwp->lmw_message;
    printf( "Thread %s: Modify thread picked entry %d of %d\n", id, i,
num_entries );

    /* Perform the modification. */
    dn = ldap_get_dn( ld, e );
    mod.mod_op = LDAP_MOD_REPLACE;
    mod.mod_type = "description";
    mod.mod_values = vals;

```

Code Example 14-4 Using POSIX Threading Under Solaris (*Continued*)

```

        printf( "Thread %s: Modifying (%s)\n", id, dn );
        rc = ldap_modify_ext_s( ld, dn, mods, NULL, NULL );
        if ( rc != LDAP_SUCCESS ) {
            rc = ldap_get_lderrno( ld, NULL, NULL );
            fprintf( stderr, "ldap_modify_ext_s: %s\n", ldap_err2string( rc ) );
        }
        free( dn );
    }
}

/* Thread for adding directory entries.
This thread randomly generates DNs for entries and attempts to add them to
the directory. */
static void *
add_thread( char *id )
{
    LDAPMod mod[5];
    LDAPMod *mods[6];
    char dn[BUFSIZ], name[40];
    char *cnvals[2], *snvals[2], *ocvals[3];
    int i, rc;

    printf( "Starting add_thread %s.\n", id );
    tsd_setup();

    /* Set up the entry to be added. */
    for ( i = 0; i < 5; i++ ) {
        mods[i] = &mod[i];
    }
    mods[5] = NULL;
    mod[0].mod_op = 0;
    mod[0].mod_type = "cn";
    mod[0].mod_values = cnvals;
    cnvals[1] = NULL;
    mod[1].mod_op = 0;
    mod[1].mod_type = "sn";
    mod[1].mod_values = snvals;
    snvals[1] = NULL;
    mod[2].mod_op = 0;
    mod[2].mod_type = "objectclass";
    mod[2].mod_values = ocvals;
    ocvals[0] = "top";
    ocvals[1] = "person";
    ocvals[2] = NULL;
    mods[3] = NULL;

    /* Randomly generate DNs and add entries. */
    for ( ;; ) {
        sprintf( name, "%d", rand() );
        sprintf( dn, "cn=%s, " BASE, name );
        cnvals[0] = name;
        snvals[0] = name;
        printf( "Thread %s: Adding entry (%s)\n", id, dn );
        rc = ldap_add_ext_s( ld, dn, mods, NULL, NULL );
        if ( rc != LDAP_SUCCESS ) {

```

Code Example 14-4 Using POSIX Threading Under Solaris (*Continued*)

```

        rc = ldap_get_lderrno( ld, NULL, NULL );
        fprintf( stderr, "ldap_add_ext_s: %s\n", ldap_err2string( rc ) );
    }
}

/* Thread for deleting directory entries.
This thread randomly selects entries for deletion. */
static void *
delete_thread( char *id )
{
    LDAPMessage *res;
    char    dn[BUFSIZ], name[40];

    printf( "Starting delete_thread %s.\n", id );
    tsd_setup();

    /* Randomly select entries for deletion. */
    for ( ;; ) {
        sprintf( name, "%d", rand() );
        sprintf( dn, "cn=%s, " BASE, name );
        printf( "Thread %s: Deleting entry (%s)\n", id, dn );
        if ( ldap_delete_ext_s( ld, dn, NULL, NULL ) != LDAP_SUCCESS ) {
            ldap_perror( ld, "ldap_delete_ext_s" );
        }
    }
}

/* Function for allocating a mutex. */
static void *
my_mutex_alloc( void )
{
    pthread_mutex_t *mutexp;
    if ( (mutexp = malloc( sizeof(pthread_mutex_t) )) != NULL ) {
        pthread_mutex_init( mutexp, NULL );
    }
    return( mutexp );
}

/* Function for freeing a mutex. */
static void
my_mutex_free( void *mutexp )
{
    pthread_mutex_destroy( (pthread_mutex_t *) mutexp );
    free( mutexp );
}

/* Error structure. */
struct ldap_error {
    int    le_errno;
    char   *le_matched;
    char   *le_errmsg;
};

/* Function to set up thread-specific data. */

```

Code Example 14-4 Using POSIX Threading Under Solaris (*Continued*)

```

static void
tsd_setup()
{
    void *tsd;
    tsd = pthread_getspecific( key );
    if ( tsd != NULL ) {
        fprintf( stderr, "tsd non-null!\n" );
        pthread_exit( NULL );
    }
    tsd = (void *) calloc( 1, sizeof(struct ldap_error) );
    pthread_setspecific( key, tsd );
}

/* Function for setting an LDAP error. */
static void
set_ld_error( int err, char *matched, char *errmsg, void *dummy )
{
    struct ldap_error *le;
    le = pthread_getspecific( key );
    le->le_errno = err;
    if ( le->le_matched != NULL ) {
        ldap_memfree( le->le_matched );
    }
    le->le_matched = matched;
    if ( le->le_errmsg != NULL ) {
        ldap_memfree( le->le_errmsg );
    }
    le->le_errmsg = errmsg;
}

/* Function for getting an LDAP error. */
static int
get_ld_error( char **matched, char **errmsg, void *dummy )
{
    struct ldap_error *le;
    le = pthread_getspecific( key );
    if ( matched != NULL ) {
        *matched = le->le_matched;
    }
    if ( errmsg != NULL ) {
        *errmsg = le->le_errmsg;
    }
    return( le->le_errno );
}

/* Function for setting errno. */
static void
set_errno( int err )
{
    errno = err;
}

/* Function for getting errno. */
static int
get_errno( void )

```


Code Example 14-4 Using POSIX Threading Under Solaris (*Continued*)

```
{  
    return( errno );  
}
```


Extended Operations

The Sun™ Java System LDAP SDK for C contains the functionality to integrate Lightweight Directory Access Protocol (LDAP) v3 extended operations. This chapter explains how LDAPv3 extended operations work and how to use those that are supported by your LDAP server. It contains the following sections:

- How Extended Operations Work
- Determining Supported Extended Operations
- Performing an Extended Operation
- Sample Code to Perform an Extended Operation

How Extended Operations Work

Extended operations are part of the LDAPv3. Each extended operation is identified by an object identifier (OID). LDAP clients can request the operation by sending an extended operation request. Within the request, the client specifies:

- The OID of the extended operation that should be performed.
- Data specific to the extended operation.

The server receives the request, performs the extended operation, and sends back a response to the client containing an OID and any additional data.

NOTE

In order to use extended operations, both the server and the client must understand the specific extended operation to be performed. From the LDAP server perspective, Sun Java System Directory Server supports a server plug-in interface that you can use to add support for extended operations. For details, see the *Sun ONE Directory Server Plug-In Programming Guide*.

Determining Supported Extended Operations

To determine the extended operations supported by the server, get the root DSE of the server, and check the `supportedExtension` attribute. The values of this attribute are the OIDs of the extended operations supported by this server. If the root DSE does not have a `supportedExtension` attribute, the server does not support any extended operations. For information on the root DSE, see “Getting the Root DSE” on page 212 of Chapter 10, “Retrieving Server Information.”

Performing an Extended Operation

To perform an extended operation, call one of the following:

- The synchronous `ldap_extended_operation_s()` function.
- The asynchronous `ldap_extended_operation()` function.

Both of these functions allow you to specify the OID of the extended operation and the data that you want applied to the operation.

TIP	Before calling the function to perform a LDAP extended operation, make sure to specify that your client is using version 3 of the LDAP. If you do not, an <code>LDAP_NOT_SUPPORTED</code> result code is returned. For details, see “Specifying the LDAP Version of Your Client” on page 65 of Chapter 3, “Writing an LDAP Client.”

Synchronous Extended Operation

If you want to wait for the results of an LDAP extended operation to complete before continuing, call the synchronous `ldap_extended_operation_s()` function. This function sends a SASL bind request to the server and blocks other work until the server sends the results of the operation back to your client.

`ldap_extended_operation_s()` returns `LDAP_SUCCESS` if the operation completed successfully or an error code if a problem occurred. See the documentation of the `ldap_extended_operation_s()` function for a list of the possible result codes.

Asynchronous Extended Operation

If you want to perform other work (in parallel) while waiting for an LDAP extended operation to complete, do the following:

1. Call the asynchronous `ldap_extended_operation()` function to send an LDAP extended operation request.

This function returns an `LDAP_SUCCESS` result code if the request was successfully sent (or an LDAP result code if an error occurred while sending the request). The function also sets the `msgidp` argument to point to a message ID identifying the extended operation. To determine whether the server sent a response to your client for this operation, call the `ldap_result()` function and pass in this message ID. The function passes back the response in an `LDAPMessage` structure.

2. Call the `ldap_parse_extended_result()` function to parse the `LDAPMessage` structure and retrieve information from the server's response.

If the server sent an OID of an extended operation to your client, the OID is passed back as the `retoidp` argument. If the server sent a data to your client, the data is specified in the `berval` structure passed back as the `retdatap` argument.

3. Call the `ldap_get_lderrno()` function to get the LDAP result code for the operation.

The function returns an `LDAP_SUCCESS` result code if the extended operation was performed successfully or an LDAP error code if a problem occurred. See the documentation of the `ldap_extended_operation()` function for a list of result codes that the server can return for this operation.

Sample Code to Perform an Extended Operation

Code Example 15-1 is an example of an LDAP client that request an extended operation from the server with the OID 1.2.3.4.

Code Example 15-1 Requesting an Extended Operation

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "ldap.h"

/* Name and port of the LDAP server you want to connect to. */
#define MY_HOST "localhost"
#define MY_PORT 389
/* DN of user (and password of user) who you want to authenticate as */
#define MGR_DN "cn=Directory Manager"
#define MGR_PW "23skidoo"
```

Code Example 15-1 Requesting an Extended Operation (*Continued*)

```

int
main( int argc, char **argv )
{
    /* OID of the extended operation that you are requesting */
    const char    *oidrequest = "1.2.3.4";
    char          *oidresult;
    struct berval  valrequest;
    struct berval  *valresult;
    LDAP          *ld;
    int rc, version;
    /* Set up the value that you want to pass to the server */
    printf( "Setting up value to pass to server...\n" );
    valrequest.bv_val = "My Value";
    valrequest.bv_len = strlen( "My Value" );
    /* Get a handle to an LDAP connection */
    printf( "Getting the handle to the LDAP connection...\n" );
    if ( ( ld = ldap_init( MY_HOST, MY_PORT ) ) == NULL ) {
        perror( "ldap_init" );
        ldap_unbind( ld );
        return( 1 );
    }
    /* Set the LDAP protocol version supported by the client
       to 3. (By default, this is set to 2. Extended operations
       are part of version 3 of the LDAP protocol.) */
    ldap_get_option( ld, LDAP_OPT_PROTOCOL_VERSION, &version );
    printf( "Resetting version %d to 3.0...\n", version );
    version = LDAP_VERSION3;
    ldap_set_option( ld, LDAP_OPT_PROTOCOL_VERSION, &version );
    /* Authenticate to the directory as the Directory Manager */
    printf( "Binding to the directory...\n" );
    if ( ldap_simple_bind_s( ld, MGR_DN, MGR_PW ) != LDAP_SUCCESS ) {
        ldap_perror( ld, "ldap_simple_bind_s" );
        ldap_unbind( ld );
        return( 1 );
    }
    /* Initiate the extended operation */
    printf( "Initiating the extended operation...\n" );
    if ( ( rc = ldap_extended_operation_s( ld, oidrequest, &valrequest,
    NULL, NULL, &oidresult, &valresult ) ) != LDAP_SUCCESS ) {
        ldap_perror( ld, "ldap_extended_operation failed: " );
        ldap_unbind( ld );
        return( 1 );
    }
    /* Get the OID and the value from the result returned by the server. */
    printf( "Operation successful.\n" );
    printf( "\tReturned OID: %s\n", oidresult );
    printf( "\tReturned value: %s\n", valresult->bv_val );
    /* Disconnect from the server. */
    ldap_unbind( ld );
    return 0;
}

```

Reference Guide

Chapter 16, “Data Type Reference”

Chapter 17, “Function Reference”

Chapter 18, “Result Codes”

Data Type Reference

This chapter contains reference material for the public data types and structures of the Sun™ Java System LDAP SDK for C. The first sections detail the data types of the grouped into task categories. The last section is an alphabetical listing of the same information. The chapter contains the following sections:

- Conventions
- Structure Summary by Task
- Structures Alphabetically

Conventions

The following sections detail certain conventions and concepts used in the LDAP SDK for C.

Typrographical

Table 16-1 lists the typographical conventions used in the names of data types and structures.

Table 16-1 Typographical Conventions in Data Types and Structures

Convention	Description of Use
LeadingCaps	Data structures needed to pass values to and from functions of the LDAP SDK for C. For consistency, use this typedef name for these structures, not their literal struct name.
ALL_CAPS	Names of callback function prototypes which you can implement for extended funtionality in your client application.

Table 16-1 Typographical Conventions in Data Types and Structures *(Continued)*

Convention	Description of Use
<code>_fns</code> suffix	Structures which hold the function pointers for callback functions, grouped by task.

Deprecated Structures

A deprecated API is one that we recommend you no longer use. While deprecated APIs are currently still implemented, they may be removed in future versions of the LDAP SDK for C. Applications should not rely on a deprecated structure as it can and will disappear without notice.

Structure Summary by Task

In the following sections, the structures defined by the LDAP SDK for C are grouped into task categories.

BER Structures

The structures listed in Table 16-2 represent data encoded using the Basic Encoding Rules (BER). The `lber.h` header file contains many other type definitions; the ones listed here are those likely to be handled directly. All others are used internally by the various functions of the LDAP SDK for C.

Table 16-2 BER Structures

Structure	Description
<code>berval</code>	<code>berval</code> represents binary data encoded using BER.
<code>BerElement</code>	<code>BerElement</code> indicates the current position during a traversal of an attribute list.

Structures for the Core API

The structures listed in Table 16-3 are used by the core functions of the LDAP SDK for C. These structures are used directly as argument or return types by functions that provide the core functionality.

Table 16-3 Core API Structures

Structure	Description
LDAP	LDAP represents a connection handle to the LDAP server.
LDAPMessage	LDAPMessage represents the results of an LDAP operation, a chain of search results, an entry in the search results, or a search reference in the search results.
LDAPMod	LDAPMod specifies changes to an attribute in an directory entry.
LDAPControl	LDAPControl represents a client or server control associated with an LDAP operation.
LDAPAPIInfo	LDAPAPIInfo represents information about the version of the LDAP SDK for C being implemented
LDAPAPIFeatureInfo	LDAPAPIFeatureInfo represents information about the extended features.

Structures for API Extensions

The structures listed in Table 16-4 are needed for extensions to the standard API within the LDAP SDK for C. These structures are used directly as arguments or return types by the functions that provide the extended functionality.

Table 16-4 API Extension Structures

Structure	Description
LDAPsortkey	LDAPsortkey represents a server control used to specify that the server should sort the search results before sending them back to the client.
LDAPVirtualList	LDAPVirtualList specifies the information that can be used to create a <i>virtual list view</i> control.
LDAPURLDesc	LDAPURLDesc represents the components of an LDAP URL.
LDAPFiltInfo	LDAPFiltInfo represents information about a filter in a filter configuration file.
LDAPFiltDesc	LDAPFiltDesc is a type of structure returned when you call <code>ldap_init_getfilter()</code> to load a filter configuration file.
FriendlyMap	FriendlyMap represents the mapping between a list of standard attribute names and their user friendly counterparts.
LDAPMemCache	LDAPMemCache represents an in-memory, client-side cache.

NOTE	Many other types, especially prototypes for callback functions, are associated with controls. These are listed separately in the subsequent sections of this chapter. The concept of controls themselves are defined and discussed in Chapter 13, “Working with LDAP Controls.”

Referral Binding Callback

This prototype for a callback function will retrieve authentication information when automatically following referrals to other servers.

Table 16-5 Referral Binding Data Type

Data Type	Description
LDAP_REBINDPROC_CALLBACK	Retrieves authentication information when following referrals to other servers.

Client-Side Sorting Callbacks

The prototype structures listed in Table 16-6 are extensions to the LDAP SDK for C that implement client side sorting of entries.

Table 16-6 Client-side Sorting Structures

Data Type	Description
LDAP_KEYGEN_CALLBACK	Function prototype that generates the sorting key for each entry to be sorted, usually by extracting a value out of an entry.
LDAP_KEYCMP_CALLBACK	Function prototype to compare 2 keys (for ordering).
LDAP_KEYFREE_CALLBACK	Function prototype that frees the memory allocated during the KEYGEN callback.
LDAP_CMP_CALLBACK	Function prototype to sort a specified set of entries.
LDAP_VALCMP_CALLBACK	Function prototype to sort a specified set of values.

Extended I/O Control

Table 16-7 lists the prototypes and structures used in the extended I/O control, an extension to the standard API.

Table 16-7 Structures and Prototypes for Extended I/O Functionality

Data Type	Description
<code>ldap_x_ext_io_fns</code>	This structure holds extended I/O function pointers.
<code>LDAP_X_POLLFD</code>	An LDAP <code>poll()</code> -like file descriptor.
<code>LDAP_X_EXTIOF_CONNECT_CALLBACK</code>	Function prototype for a callback that opens a socket connection.
<code>LDAP_X_EXTIOF_CLOSE_CALLBACK</code>	Function prototype for a callback that closes a socket connection.
<code>LDAP_X_EXTIOF_POLL_CALLBACK</code>	Function prototype for a callback that surveys a server about a particular event.
<code>LDAP_X_EXTIOF_NEWHANDLE_CALLBACK</code>	Function prototype for a callback that defines a new session handle.
<code>LDAP_X_EXTIOF_DISPOSEHANDLE_CALLBACK</code>	Function prototype for a callback that disposes of a session handle.
<code>ldap_x_hostlist_status</code>	This structure holds utility functions for parsing space-separated host lists.

Memory Management Control

The prototypes and structures listed in Table 16-8 are used in memory management control, an extension to the standard API. They allow developers to specify their own memory allocation mechanisms and functions.

Table 16-8 Memory Management Datatypes

Data Type	Description
<code>ldap_memalloc_fns</code>	This structure holds the memory allocation callback functions.
<code>LDAP_MALLOC_CALLBACK</code> <code>LDAP_CALLOC_CALLBACK</code> <code>LDAP_REALLOC_CALLBACK</code> <code>LDAP_FREE_CALLBACK</code>	These callbacks allow developers to specify their own memory allocation mechanisms.

Thread Signaling Controls

The prototypes and structures listed in Table 16-9 are used in the thread signaling control, an extension to the standard API.

Table 16-9 Thread Signaling Control Data Types

Data Type	Description
ldap_thread_fns	Structure that contains a set of pointers to functions you want to use when writing a multithreaded client.
LDAP_TF_MUTEX_ALLOC_CALLBACK	Function prototype for allocating a mutex.
LDAP_TF_MUTEX_FREE_CALLBACK	Function prototype freeing a mutex.
LDAP_TF_MUTEX_LOCK_CALLBACK	Function prototype for locking critical sections of code.
LDAP_TF_MUTEX_UNLOCK_CALLBACK	Function prototype for unlocking critical sections of code.
LDAP_TF_GET_ERRNO_CALLBACK	Function prototype for getting the value of the errno variable.
LDAP_TF_SET_ERRNO_CALLBACK	Function prototype for setting the value of the errno variable.
LDAP_TF_GET_LDERRNO_CALLBACK	Function prototype for getting error values from calls to functions in the libldap library.
LDAP_TF_SET_LDERRNO_CALLBACK	Function prototype for setting error values from calls to functions in the libldap library.
ldap_extra_thread_fns	Structure that contains a set of pointers to additional functions you want to use when writing a multithreaded client.
LDAP_TF_MUTEX_TRYLOCK_CALLBACK	Function prototype for attempting to lock a mutex.
LDAP_TF_SEMA_ALLOC_CALLBACK	Function prototype for allocating a semaphore.
LDAP_TF_SEMA_FREE_CALLBACK	Function prototype for freeing a semaphore.
LDAP_TF_SEMA_WAIT_CALLBACK	Function prototype for waiting for the value of a semaphore to be greater than 0.
LDAP_TF_SEMA_POST_CALLBACK	Function prototype for incrementing the value of a semaphore.
LDAP_TF_THREADID_CALLBACK	Function prototype to return an identifier that is unique to the calling thread.

Deprecated and Outdated Types

Deprecated types are those being dropped from the standard or from the extensions to the LDAP SDK for C. They are not guaranteed to be defined in future versions of the API. However, all of these types are still defined in this version for backwards compatibility. Table 16-10 lists the deprecated prototypes.

Table 16-10 Deprecated Data Types and Their Replacements

Data Type	Replacement
LDAP_CANCELPROC_CALLBACK	Provides a way to be cancel a process, for example, by a user or because some other condition occurs.
ldap_cache_fns	ldap_cache_fns is a deprecated structure and the typedef declarations associated with it have also been deprecated.
LDAP_CF_BIND_CALLBACK	
LDAP_CF_UNBIND_CALLBACK	
LDAP_CF_SEARCH_CALLBACK	
LDAP_CF_COMPARE_CALLBACK	
LDAP_CF_ADD_CALLBACK	
LDAP_CF_DELETE_CALLBACK	
LDAP_CF_MODIFY_CALLBACK	
LDAP_CF_MODRDN_CALLBACK	
LDAP_CF_RESULT_CALLBACK	
LDAP_CF_FLUSH_CALLBACK	
LDAPVersion	Replaced by the LDAPAPIInfo type which is returned by the ldap_get_option() function using the LDAP_OPT_API_INFO option.

While not officially deprecated, other data types are outdated because they have superseded by new types. Outdated types may become deprecated in future releases of the LDAP SDK for C. Table 16-11 lists the outdated types and the newer types which implement the same functionality.

Table 16-11 Outdated Types and Their New Equivalent

Old Data Type	New, Equivalent Data Type
LDAPHostEnt	The DNS resolver callbacks are an outdated extension to the API. The new way of specifying host information is through the LDAP_X_EXTIOF_CONNECT_CALLBACK prototype.
LDAP_DNSFN_GETHOSTBYNAME	
LDAP_DNSFN_GETHOSTBYADDR	
ldap_dns_fns	

Table 16-11 Outdated Types and Their New Equivalent (*Continued*)

Old Data Type	New, Equivalent Data Type
LDAP_IOF_CLOSE_CALLBACK	Replaced by the types for extended I/O functionality detailed in Table 16-7 on page 277.
LDAP_IOF_CONNECT_CALLBACK	
LDAP_IOF_IOCTL_CALLBACK	
LDAP_IOF_READ_CALLBACK	
LDAP_IOF_SELECT_CALLBACK	
LDAP_IOF_SOCKET_CALLBACK	
LDAP_IOF_SSL_ENABLE_CALLBACK	
LDAP_IOF_WRITE_CALLBACK	
ldap_io_fns	

Structures Alphabetically

The following sections detail the structures and controls of the LDAP SDK for C in alphabetical order.

berval

`berval` is a structure that represents binary data encoded using simplified Basic Encoding Rules (BER).

Description

Use a `berval` structure when working with attributes that contain binary data (such as a graphic or audio file). The data and the size of the data are both included in a `berval` structure. Code Example 16-1 illustrates the definition of the `berval` structure.

Code Example 16-1 `berval` Definition

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

The fields in the `berval` structure are described in Table 16-12.

Table 16-12 berval Field Descriptions

Field	What It Contains
<code>bv_len</code>	The length of the data in bytes.
<code>bv_val</code>	A pointer to the binary data itself.

BerElement

The `BerElement` structure represents data encoded using the Basic Encoding Rules (BER).

Description

You use this opaque data type to keep track of the current attribute during the traversal of an attribute list. Calling the `ldap_first_attribute()` function allocates memory for a `BerElement` structure and initializes it to select the values of the first sttribute in the entry. Subsequently, `ldap_next_attribute()` is called to obtain the name of each remaining attribute and set `BerElement` to select their values. Once `BerElement` has been set to select an attribute's values, `ldap_get_values()` can be used to obtain all of them from the buffer. When you are done reading the attributes, you need to free the `BerElement` structure from memory by calling the `ldap_ber_free()` function. (A `BerElement` structure can also be allocated by calling the `ber_alloc_t()` or the `ber_init()` function. In these cases, free the memory allocated to the `BerElement` structure by using the `ber_free()` function.)

NOTE	The <code>BerElement</code> definition is not completely exposed in <code>lber.h</code> because the fields within the structure are not intended to be accessible to clients.

FriendlyMap

`FriendlyMap` represents the mapping between a list of standard attribute names and their user-friendly counterparts. For example, you can represent the list of two-letter state codes (CA, IA) with their corresponding state names (California, Iowa), or map Country ISO codes to the full country names, in a `FriendlyMap` structure.

NOTE	<code>FriendlyMap</code> is not completely defined in <code>ldap-extension.h</code> because the fields within the structure are not intended to be accessible to clients.

Related Functions

- `ldap_friendly_name()` allocates a `FriendlyMap` structure and reads a list of of standard attribute names and their user friendly counterparts from a file.
- `ldap_free_friendlymap()` frees the memory allocated for a `FriendlyMap` structure.

LDAP

LDAP is an opaque data type representing a connection with the LDAP server. It is initialized through a call to either the `ldap_init()` or `ldapssl_init()` function. LDAP maintains the state of an LDAP session for the duration of the connection. When you call functions that perform LDAP operations on an LDAP server (for example, `ldap_search_ext()` to search the directory or `ldap_modify_ext()` to update an entry), you need to pass a pointer to this connection handle. With the LDAP structure, you can also:

- Call the `ldap_get_option()` and `ldap_set_option()` functions to view or modify the properties of the connection.
- Call the `ldap_unbind()` or `ldap_unbind_s()` function to close the connection and free the LDAP structure.

NOTE

LDAP is not completely defined in `ldap-standard.h` because the fields within the structure are not intended to be accessible to clients.

LDAPAPIFeatureInfo

`LDAPAPIFeatureInfo` is a structure that represents information about the extended features of the LDAP SDK for C. Code Example 16-2 illustrates the definition of the `LDAPAPIFeatureInfo` structure.

Code Example 16-2 LDAPAPIFeatureInfo Definition

```
#define LDAP_FEATURE_INFO_VERSION
typedef struct ldap_apifeature_info {
    int    ldapaif_info_version;    /* version of this struct (1) */
    char   *ldapaif_name;           /* name of supported feature */
    int    ldapaif_version;         /* revision of supported feature */
} LDAPAPIFeatureInfo;
```

The LDAPAPIFeatureInfo structure can be retrieved by using a sequence like the one displayed in Code Example 16-3.

Code Example 16-3 Sequence to Retrieve LDAPAPIFeatureInfo

```
LDAPAPIFeatureInfo ldfi;
ldfi.ldapaif_info_version = LDAP_FEATURE_INFO_VERSION;
ldfi.ldapaif_name = "VIRTUAL_LIST_VIEW";
if ( ldap_get_option( NULL, LDAP_OPT_API_FEATURE_INFO, &ldfi ) == 0 ) ...
```

Parameters

This structure has the parameters listed in Table 16-13.

Table 16-13 LDAPAPIFeatureInfo Structure Parameters

Parameter	Description
ldapaif_info_version	Specifies the version number of the LDAPAPIFeatureInfo structure. This must be set to LDAP_FEATURE_INFO_VERSION before call to ldap_get_option() is performed.
ldapaif_name	Pointer to NULL-terminated string that specifies the name of the supported feature.
ldapaif_version	Specifies the version number of the supported feature.

LDAPAPIInfo

LDAPAPIInfo is a structure that represents information about the API and supported extensions. Code Example 16-4 illustrates the definition of the LDAPAPIInfo structure.

Code Example 16-4 LDAPAPIInfo Definition

```
#define LDAP_API_INFO_VERSION
typedef struct ldapapiinfo {
    int ldapai_info_version; /* version of LDAPAPIInfo (1) */
    int ldapai_api_version; /* revision of API supported */
    int ldapai_protocol_version; /* highest LDAP version supported */
    char **ldapai_extensions; /* names of API extensions */
    char *ldapi_vendor_name; /* name of supplier */
}
```

Code Example 16-4 LDAPAPIInfo Definition

```
int ldapai_vendor_version; /* supplier-specific version times 100
*/
} LDAPAPIInfo;
```

The LDAPAPIInfo structure can be retrieved by using a sequence like the one displayed in Code Example 16-5.

Code Example 16-5 Sequence to Retrieve LDAPAPIInfo

```
LDAPAPIInfo ldai;
ldai.ldapai_info_version = LDAP_API_INFO_VERSION;
if ( ldap_get_option( NULL, LDAP_OPT_API_INFO, &ldia ) == 0 ) ...
```

Parameters

This structure has the parameters listed in Table 16-14.

Table 16-14 LDAPAPIInfo Structure Parameters

Parameter	Description
ldapai_info_version	Specifies the version number of the LDAPAPIInfo structure. This must be set to LDAP_API_INFO_VERSION before a call to ldap_get_option() is performed.
ldapai_api_version	Specifies the version number of the API that is supported.
ldapai_protocol_version	Specifies the latest LDAP version supported by the LDAP library.
ldapai_extensions	Points to a NULL-terminated array of character strings that names the supported LDAP extensions. If none are supported, this field is set to NULL. The application is responsible for freeing this memory by calling the ldap_value_free() function.
ldapai_vendor_name	Pointer to a NULL-terminated string that contains the vendor's name. Call the ldap_memfree() function to free the memory.
ldapai_vendor_version	Specifies the vendor's version of the LDAP libraries.

ldap_cache_fns

ldap_cache_fns is a deprecated structure. The following are also deprecated:

- LDAP_CF_ADD_CALLBACK
- LDAP_CF_BIND_CALLBACK
- LDAP_CF_COMPARE_CALLBACK
- LDAP_CF_DELETE_CALLBACK
- LDAP_CF_FLUSH_CALLBACK
- LDAP_CF_MODIFY_CALLBACK
- LDAP_CF_MODRDN_CALLBACK
- LDAP_CF_RESULT_CALLBACK
- LDAP_CF_SEARCH_CALLBACK
- LDAP_CF_UNBIND_CALLBACK

See Also

Please see the `ldap_memcache_*` functions in Chapter 17, “Function Reference” for current usage.

LDAP_CALLOC_CALLBACK

This callback function prototype of `ldap_memalloc_fns` represents memory allocation.

See Also

`ldap_memalloc_fns`

LDAP_CANCELPROC_CALLBACK

This callback function prototype is to-be-deprecated.

See Also

`ldap_ufn_search_c()`, `ldap_ufn_search_ct()`

LDAP_CF_ADD_CALLBACK

This callback function prototype of `ldap_cache_fns` is deprecated.

LDAP_CF_BIND_CALLBACK

This callback function prototype of `ldap_cache_fns` is deprecated.

LDAP_CF_COMPARE_CALLBACK

This callback function prototype of `ldap_cache_fns` is deprecated.

LDAP_CF_DELETE_CALLBACK

This callback function prototype of `ldap_cache_fns` is deprecated.

LDAP_CF_FLUSH_CALLBACK

This callback function prototype of `ldap_cache_fns` is deprecated.

LDAP_CF_MODIFY_CALLBACK

This callback function prototype of `ldap_cache_fns` is deprecated.

LDAP_CF_MODRDN_CALLBACK

This callback function prototype of `ldap_cache_fns` is deprecated.

LDAP_CF_RESULT_CALLBACK

This callback function prototype of `ldap_cache_fns` is deprecated.

LDAP_CF_SEARCH_CALLBACK

This callback function prototype of `ldap_cache_fns` is deprecated.

LDAP_CF_UNBIND_CALLBACK

This callback function prototype of `ldap_cache_fns` is deprecated.

LDAP_CMP_CALLBACK

LDAP_CMP_CALLBACK specifies the prototype for a comparison callback function used when sorting values. If you define a function with this prototype and specify it when calling `ldap_sort_entries()` or `ldap_multisort_entries()`, it will be called by your LDAP client to sort a specified set of entries. Code Example 16-6 is the prototype.

Code Example 16-6 LDAP_CMP_CALLBACK Prototype

```
typedef int (LDAP_C LDAP_CALLBACK
LDAP_CMP_CALLBACK)(const char*, const char*);
```

See Also

`ldap_sort_entries()`, `ldap_multisort_entries()`.

LDAPControl

LDAPControl represents a client or server control associated with an LDAP operation. If set as a server control, it is sent to the server along with operation requests; if set as a client control, it is interpreted locally by the client.

NOTE

Controls are part of the LDAPv3. You can use a control to extend the functionality of an LDAP operation. There are two basic types of controls described in the LDAPv3:

- *Server controls* are controls that are sent from the client to the server along with an LDAP request. (In some cases, a server can include a control in the response it sends back to the client.) For example, you can include a server control in a search request to specify that you want the server to sort the search results before sending them back.
- *Client controls* are controls that can extend the client but are never sent to the server. As a general example, you might be able to pass a client control to an LDAP API function, which might parse the control and use the data that you've specified in the control.

Sun Java System LDAP SDK for C does not currently support any client controls. For more information on LDAP controls, see Chapter 13, "Working with LDAP Controls."

Code Example 16-7 illustrates how LDAPControl is defined.

Code Example 16-7 Definition of LDAPControl

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

Table 16-15 describes the fields in this structure.

Table 16-15 LDAPControl Field Descriptions

Field	What It Contains
ldctl_oid	Object identifier (OID) of the control.
ldctl_value	berval structure containing data associated with the control. If you want to specify a zero-length value, set ldctl_value.bv_len to 0 and ldctl_value.bv_val to a zero-length string. To indicate that no data is associated with the control, set ldctl_value.bv_val to NULL.
ldctl_iscritical	Specifies whether or not the control is critical to the operation. This field can have one of the following values: • A non-zero value specifies that the control is critical to the operation. • 0 specifies that the control is not critical to the operation.

LDAP_DNSFN_GETHOSTBYADDR

LDAP_DNSFN_GETHOSTBYADDR specifies the prototype for a callback function equivalent to the gethostbyaddr_r() function available on some UNIX® platforms.

CAUTION

While not officially deprecated, the functionality of this callback prototype is superseded by the extended I/O functions. The new way of specifying host information is through the LDAP_X_EXTIOF_CONNECT_CALLBACK prototype.

Description

If you define a function with this prototype and set it in the ldap_dns_fns structure, your function will be called by the SDK on behalf of your LDAP client if it needs to get the host name of the LDAP server to which it is connected.

NOTE In this version of the LDAP SDK for C, the functions of the API never call this function, and your implementation will never be executed. Therefore, its implementation is optional.

Code Example 16-8 is the prototype specified by this data type.

Code Example 16-8 LDAP_DNSFN_GETHOSTBYADDR Prototype

```
typedef LDAPHostEnt * (LDAP_C LDAP_CALLBACK
LDAP_DNSFN_GETHOSTBYADDR)( const char *addr, int length, int type,
LDAPHostEnt *result, char *buffer, int buflen, int *statusp,
void *extradata );
```

LDAP_DNSFN_GETHOSTBYNAME

LDAP_DNSFN_GETHOSTBYNAME specifies the prototype for a callback function equivalent to the `gethostbyname_r()` function available on some UNIX platforms.

CAUTION While not officially deprecated, the functionality of this callback prototype is superseded by the extended I/O functions. The new way of specifying host information is through the `LDAP_X_EXTIOF_CONNECT_CALLBACK` prototype.

Description

If you define a function with this prototype and set it in the `ldap_dns_fns` structure, your function will be called by the SDK on behalf of your LDAP client to get the host entry for the LDAP server when connecting to the server. Code Example 16-9 is the prototype specified by this data type.

Code Example 16-9 LDAP_DNSFN_GETHOSTBYNAME Prototype

```
typedef LDAPHostEnt * (LDAP_C LDAP_CALLBACK
LDAP_DNSFN_GETHOSTBYNAME)( const char *addr, int length, int type,
LDAPHostEnt *result, char *buffer, int buflen, int *statusp,
void *extradata );
```

ldap_dns_fns

ldap_dns_fns contains a set of pointers to DNS functions (equivalent to the gethostbyname_r() and gethostbyaddr_r() functions available on some UNIX platforms).

CAUTION

While not officially deprecated, the functionality of this structure is superceded by the extended I/O functions. The new way of specifying host information is through the LDAP_X_EXTIOF_CONNECT_CALLBACK prototype.

Description

You can use this structure if you want the SDK to call these functions when looking up the host name or IP address for the LDAP server. For example, you could use this to call versions of the DNS functions that are safe for use in a multi-threaded application. The function must have the prototype specified by LDAP_DNSFN_GETHOSTBYADDR. If NULL, the standard built-in OS routine is used.

After you set the fields in this structure, you can register the functions for use by calling the ldap_set_option() function and setting the LDAP_OPT_DNS_FN_PTRS option to this structure. Code Example 16-10 illustrates how ldap_dns_fns is defined.

Code Example 16-10 ldap_dns_fns Definition

```
struct ldap_dns_fns {
    void *lddnsfn_extradata;
    int lddnsfn_bufsize;
    LDAP_DNSFN_GETHOSTBYNAME *lddnsfn_gethostbyname;
    LDAP_DNSFN_GETHOSTBYADDR *lddnsfn_gethostbyaddr;
};
```

Table 16-16 describes the fields in this structure.

Table 16-16 ldap_dns_fns Field Descriptions

Field	What It Contains
lddnsfn_extradata	Value passed in the extradata argument of the LDAP_DNSFN_GETHOSTBYADDR and LDAP_DNSFN_GETHOSTBYNAME function calls.

Table 16-16 ldap_dns_fns Field Descriptions (*Continued*)

Field	What It Contains
lddnsfn_bufsize	Specifies the size of the buffer that you want passed to your DNS callback function. Your LDAP client passes this value as the buf len argument of the LDAP_DNSFN_GETHOSTBYADDR and LDAP_DNSFN_GETHOSTBYNAME function calls.
lddnsfn_gethostbyname	Function pointer for getting the host entry for the LDAP server. This function is called by the client when connecting to the server if the function pointer is not NULL. The function must have the prototype specified by LDAP_DNSFN_GETHOSTBYNAME. If NULL, the standard built-in OS routine is used.
lddnsfn_gethostbyaddr	Function pointer for getting the host name of the LDAP server. This function is called by the client when needed if the function pointer is not NULL.

ldap_extra_thread_fns

The ldap_extra_thread_fns structure contains a set of pointers to additional functions that you can use when writing a multithreaded client. Your client calls these functions when getting results from the LDAP structure.

TIP	The LDAP SDK for C ignores all the elements in this structure except for the ltf_threadid_fn function. Calling ltf_threadid_fn will, in some cases, enhance the performance of a multithreaded program.

After you set the fields in this structure, you can register the functions for use by calling the ldap_set_option() function and setting the LDAP_OPT_EXTRA_THREAD_FN_PTRS option to this structure. Code Example 16-11 illustrates the definition of the ldap_extra_thread_fns structure.

Code Example 16-11 ldap_extra_thread_fns Definition

```

struct ldap_extra_thread_fns {
    LDAP_TF_MUTEX_TRYLOCK_CALLBACK *ltf_mutex_trylock;
    LDAP_TF_SEMA_ALLOC_CALLBACK *ltf_sema_alloc;
    LDAP_TF_SEMA_FREE_CALLBACK *ltf_sema_free;
    LDAP_TF_SEMA_WAIT_CALLBACK *ltf_sema_wait;
    LDAP_TF_SEMA_POST_CALLBACK *ltf_sema_post;
    LDAP_TF_THREADID_CALLBACK *ltf_threadid_fn;
};

```

Table 16-17 describes the fields in this structure.

Table 16-17 ldap_extra_thread_fns Field Descriptions

Field	What It Contains
ltf_mutex_trylock	Function pointer for attempting to lock a mutex. This function is called by the client when needed if the function pointer is not NULL. The function must have the prototype specified by LDAP_TF_MUTEX_TRYLOCK_CALLBACK.
ltf_sema_alloc	Function pointer for allocating a semaphore. This function is called by the client if needed when the function pointer is not NULL. The function must have the prototype specified by LDAP_TF_SEMA_ALLOC_CALLBACK.
ltf_sema_free	Function pointer for freeing a semaphore. This function is called by the client when needed if the function pointer is not NULL. The function must have the prototype specified by LDAP_TF_SEMA_FREE_CALLBACK.
ltf_sema_wait	Function pointer for waiting for the value of a semaphore to be greater than 0. This function is called by the client when needed if the function pointer is not NULL. The function must have the prototype specified by LDAP_TF_SEMA_WAIT_CALLBACK.
ltf_sema_post	Function pointer for incrementing the value of a semaphore. This function is called by the client when needed if the function pointer is not NULL. The function must have the prototype specified by LDAP_TF_SEMA_POST_CALLBACK.
ltf_threadid_fn	Function pointer that is called to retrieve the unique identifier for the calling thread. If this is NULL, it is not used. An example of a similar function in the POSIX threads standard is pthread_self(). The function must have the prototype specified by LDAP_TF_THREADID_CALLBACK.

For an example of setting up the ldap_extra_thread_fns structure, see Chapter 14, “Writing Multithreaded Clients.”

LDAPFiltDesc

LDAPFiltDesc is a structure that is returned when you call ldap_init_getfilter() to load a filter configuration file.

NOTE	The LDAPFiltDesc definition is not completely exposed in ldap-extension.h because the fields within the structure are not intended to be accessible to clients.

Description

After calling the `ldap_init_getfilter()` function, use the pointer to the returned `LDAPFiltDesc` structure in subsequent calls to get information about filters in the filter configuration file.

LDAPFiltInfo

`LDAPFiltInfo` represents information about a filter in the filter configuration file.

Description

When you call the `ldap_getfirstfilter()` or `ldap_getnextfilter()` functions to get a filter from the filter configuration file, they return a pointer to an `LDAPFiltInfo` structure containing the information about the filter. Code Example 16-12 illustrates how `LDAPFiltInfo` is defined.

Code Example 16-12 LDAPFiltInfo Definition

```
typedef struct ldap_filt_info {
    char *lfi_filter;
    char *lfi_desc;
    int lfi_scope;
    int lfi_isexact;
    struct ldap_filt_info *lfi_next;
} LDAPFiltInfo;
```

Table 16-18 details the fields in `LDAPFiltInfo`.

Table 16-18 LDAPFilterInfo Field Descriptions

Field	What It Contains
<code>lfi_filter</code>	The filter (for example, <code>(cn=d*)</code>).
<code>lfi_desc</code>	Description of the filter (the fifth field in the filter configuration file).

Table 16-18 LDAPFilterInfo Field Descriptions (*Continued*)

Field	What It Contains
lfi_scope	The scope of the filter (the sixth field in the filter configuration file), which can be one of the following values: • LDAP_SCOPE_BASE specifies that the search will be restricted to the current DN. • LDAP_SCOPE_ONELEVEL specifies that the search will be restricted to the entries at the level beneath the current DN. • LDAP_SCOPE_SUBTREE specifies that the search will encompass entries at all levels beneath the current DN. If the scope of the filter is not specified in the filter configuration file, the scope is LDAP_SCOPE_SUBTREE by default.
lfi_isexact	Specifies whether or not the filter is an exact filter (an exact filter contains no wildcards and does not match words that sound alike): • 0 specifies that the filter is not an exact filter. • 1 specifies that the filter is an exact filter.
lfi_next	Pointer to the LDAPFiltInfo structure representing the next filter in the filter list.

Code Example 16-13 prints out information about a filter.

Code Example 16-13 Sample LDAPFiltInfo Code

```
LDAPFiltInfo *lfip;
/* Print out the filter */
printf( "Filter:\t%s\n", lfdp->lfd_filter );
printf( "Description:\t%s\n", lfdp->lfd_desc );
```

For example, in the filter configuration file, if the first filter that applies to the value "@" is:

"@" " " "(mail=%v)" "email address is" "onelevel"

The code prints out:

Filter: (mail=@)
Description: email address is

LDAP_FREE_CALLBACK

This callback function prototype of `ldap_memalloc_fns` is used to free allocated memory.

See Also

`ldap_memalloc_fns`

LDAPHostEnt

The `LDAPHostEnt` structure represents an entry for a host found by a domain name server.

CAUTION While not officially deprecated, the functionality of this callback prototype is superseded by the extended I/O functions. The new way of specifying host information is through the `LDAP_X_EXTIOF_CONNECT_CALLBACK` prototype.

Description

This type is similar to the `hostent` structure returned by functions such as `gethostbyname_r()` on UNIX. If you are writing your own DNS functions for use by the client, they should return the host entry in this type of structure.

The fields in this structure should point to addresses within the buffer that is passed to the DNS callback (referenced in the LDAP callback function). This buffer contains the host data. The pointers in the `hostent` structure returned by the function point to the data in this buffer. Code Example 16-14 illustrates how `LDAPHostEnt` is defined.

Code Example 16-14 LDAPHostEnt Definition

```
typedef struct LDAPHostEnt {
    char *ldaphe_name;
    char **ldaphe_aliases;
    int ldaphe_addrtype;
    int ldaphe_length;
    char **ldaphe_addr_list;
} LDAPHostEnt;
```

Table 16-19 defines the fields in the `LDAPHostEnt` structure.

Table 16-19 LDAPHostEnt Field Descriptions

Field	What It Contains
ldaphe_name	Canonical name of the host
ldaphe_aliases	List of aliases for this host
ldaphe_addrtype	Address type of the host
ldaphe_length	Length of the address
ldaphe_addr_list	List of addresses for this host (as returned by the name server)

See Also

LDAP_DNSFN_GETHOSTBYADDR, LDAP_DNSFN_GETHOSTBYNAME

LDAP_IOF_CLOSE_CALLBACK

LDAP_IOF_CLOSE_CONNECT_CALLBACK specifies the prototype for a callback function equivalent to the standard close() system call.

Description

If you define a function with this prototype and set it in the ldap_io_fns structure, the function will be called by your LDAP client. Code Example 16-15 illustrates the prototype.

Code Example 16-15 LDAP_IOF_CLOSE_CONNECT_CALLBACK Prototype

```
typedef int (LDAP_C LDAP_CALLBACK
LDAP_IOF_CLOSE_CALLBACK )( LBER_SOCKET );
```

LDAP_IOF_CONNECT_CALLBACK

LDAP_IOF_CONNECT_CALLBACK specifies the prototype for a callback function equivalent to the standard connect() network I/O function.

Description

If you define a function with this prototype and set it in the ldap_io_fns structure, the function will be called by your LDAP client. Code Example 16-16 is the prototype.

Code Example 16-16 LDAP_IOF_CONNECT_CALLBACK Prototype

```
typedef int (LDAP_C LDAP_CALLBACK
LDAP_IOF_CONNECT_CALLBACK )( LBER_SOCKET,
struct sockaddr *, int );
```

LDAP_IOF_IOCTL_CALLBACK

LDAP_IOF_IOCTL_CALLBACK specifies the prototype for a callback function equivalent to the standard `ioctl()` system call.

Description

If you define a function with this prototype and set it in the `ldap_io_fns` structure, the function will be called by your LDAP client. Code Example 16-17 is the prototype.

Code Example 16-17 LDAP_IOF_IOCTL_CALLBACK Prototype

```
typedef int (LDAP_C LDAP_CALLBACK
LDAP_IOF_IOCTL_CALLBACK)( LBER_SOCKET, int, ... );
```

LDAP_IOF_READ_CALLBACK

LDAP_IOF_READ_CALLBACK specifies the prototype for a callback function equivalent to the standard `read()` I/O function.

Description

If you define a function with this prototype and set it in the `ldap_io_fns` structure, the function will be called by your LDAP client. Code Example 16-18 is the prototype.

Code Example 16-18 LDAP_IOF_READ_CALLBACK Prototype

```
typedef int (LDAP_C LDAP_CALLBACK
LDAP_IOF_READ_CALLBACK)( LBER_SOCKET, void *, int );
```

LDAP_IOF_SELECT_CALLBACK

LDAP_IOF_SELECT_CALLBACK specifies the prototype for a callback function equivalent to the standard `select()` I/O function.

Description

If you define a function with this prototype and set it in the `ldap_io_fns` structure, the function will be called by your LDAP client. Code Example 16-19 is the prototype.

Code Example 16-19 LDAP_IOF_SELECT_CALLBACK Prototype

```
typedef int (LDAP_C LDAP_CALLBACK
LDAP_IOF_SELECT_CALLBACK)( int, fd_set *, fd_set *,
    fd_set *, struct timeval * );
```

LDAP_IOF_SOCKET_CALLBACK

LDAP_IOF_SOCKET_CALLBACK specifies the prototype for a callback function equivalent to the standard `socket()` network I/O function.

Description

If you define a function with this prototype and set it in the `ldap_io_fns` structure, the function will be called by your LDAP client. Code Example 16-20 is the prototype.

Code Example 16-20 LDAP_IOF_SOCKET_CALLBACK Prototype

```
typedef LBER_SOCKET (LDAP_C LDAP_CALLBACK
LDAP_IOF_SOCKET_CALLBACK)( int, int, int );
```

LDAP_IOF_SSL_ENABLE_CALLBACK

LDAP_IOF_SSL_ENABLE_CALLBACK specifies the prototype for a callback function equivalent to the `ssl_enable()` function.

Description

If you define a function with this prototype and set it in the `ldap_io_fns` structure, the function will be called by your LDAP client. Code Example 16-21 is the prototype.

Code Example 16-21 LDAP_IOF_SSL_ENABLE_CALLBACK Prototype

```
typedef int (LDAP_C LDAP_CALLBACK
LDAP_IOF_SSL_ENABLE_CALLBACK )( LBER_SOCKET );
```

LDAP_IOF_WRITE_CALLBACK

LDAP_IOF_WRITE_CALLBACK specifies the prototype for a callback function equivalent to the standard `write()` I/O function.

Description

If you define a function with this prototype and set it in the `ldap_io_fns` structure, the function will be called by your LDAP client. Code Example 16-22 is the prototype.

Code Example 16-22 LDAP_IOF_WRITE_CALLBACK Prototype

```
typedef int (LDAP_C LDAP_CALLBACK LDAP_IOF_WRITE_CALLBACK)
( LBER_SOCKET, const void *, int );
```

ldap_io_fns

The `ldap_io_fns` structure contains a set of pointers to input/output functions that you want used with SDK. You need to set up this structure if you want to connect to the LDAP server using a Secure Sockets Layer (SSL). Code Example 16-23 illustrates the definition of the `ldap_io_fns` structure.

Code Example 16-23 ldap_io_fns Structure

```
struct ldap_io_fns {
    LDAP_IOF_READ_CALLBACK *lrof_read;
    LDAP_IOF_WRITE_CALLBACK *lrof_write;
    LDAP_IOF_SELECT_CALLBACK *lrof_select;
    LDAP_IOF_SOCKET_CALLBACK *lrof_socket;
    LDAP_IOF_IOCTL_CALLBACK *lrof_ioctl;
```

Code Example 16-23 ldap_io_fns Structure (*Continued*)

```
LDAP_IOF_CONNECT_CALLBACK *liof_connect;
LDAP_IOF_CLOSE_CALLBACK *liof_close;
LDAP_IOF_SSL_ENABLE_CALLBACK *liof_ssl_enable;
};
```

Table 16-20 defines the fields in this structure.

Table 16-20 ldap_io_fns Field Descriptions

Field	What It Contains
liof_read	Function pointer to the equivalent of the standard read() I/O function. The function must have the prototype specified by LDAP_IOF_READ_CALLBACK.
liof_write	Function pointer to the equivalent of the standard write() I/O function. The function must have the prototype specified by LDAP_IOF_WRITE_CALLBACK.
liof_select	Function pointer to the equivalent of the standard select() I/O function. The function must have the prototype specified by LDAP_IOF_SELECT_CALLBACK.
liof_socket	Function pointer to the equivalent of the standard socket() network I/O function. The function must have the prototype specified by LDAP_IOF_SOCKET_CALLBACK.
liof_ioctl	Function pointer to the equivalent of the standard ioctl() system call. The function must have the prototype specified by LDAP_IOF_IOCTL_CALLBACK.
liof_connect	Function pointer to the equivalent of the standard connect() network I/O function. The function must have the prototype specified by LDAP_IOF_CONNECT_CALLBACK.
liof_close	Function pointer to the equivalent of the standard close() system call. The function must have the prototype specified by LDAP_IOF_CLOSE_CALLBACK.
liof_ssl_enable	Function pointer to the equivalent of the ssl_enable() function. The function must have the prototype specified by LDAP_IOF_SSL_ENABLE_CALLBACK.

LDAP_KEYCMP_CALLBACK

LDAP_KEYCMP_CALLBACK specifies the prototype for a callback function to sort a specified set of entries.

Definition

If you define a function with this prototype and specify it when calling `ldap_keysort_entries()`, the function will be called by your LDAP client to sort a specified set of entries.

See Also

`ldap_keysort_entries()`

LDAP_KEYFREE_CALLBACK

`LDAP_KEYFREE_CALLBACK` specifies the prototype for a callback function that frees a sorting key.

Description

`ldap_keysort_entries()` requires a function pointer that matches the signature defined by this type.

LDAP_KEYGEN_CALLBACK

`LDAP_KEYGEN_CALLBACK` specifies the prototype for a callback function that generates a key for sorting.

Description

`ldap_keysort_entries()` requires a function pointer that matches the signature defined by this type.

LDAP_MALLOC_CALLBACK

This callback function prototype of `ldap_memalloc_fns` represents memory allocation.

See Also

`ldap_memalloc_fns`

ldap_memalloc_fns

ldap_memalloc_fns is a structure representing callback functions for memory allocation. These are global and can not be set on a per-LDAP session handle basis. Install your own functions by making a call similar to the one detailed in Code Example 16-24.

Code Example 16-24 Installing ldap_memalloc_fns

```
ldap_set_option( NULL, LDAP_OPT_MEMALLOC_FN_PTRS, &memalloc_fns )
```

See Also

- LDAP_MALLOC_CALLBACK
- LDAP_CALLOC_CALLBACK
- LDAP_REALLOC_CALLBACK
- LDAP_FREE_CALLBACK

LDAPMemCache

LDAPMemCache is a type of structure representing an in-memory, client-side cache. You can create a cache and specify the following information:

- The maximum size of the cache.
- The maximum amount of time to keep an item in the cache.
- A set of base DN's for the search requests that you want to cache (optional).
- A set of functions to ensure the thread-safety of the cache.

To use a cache, you need to associate it with a connection handle (and LDAP structure). Before a search request is sent to the server, the cache is checked to determine if the same request was made before. If an earlier request was cached, the search results are retrieved from the cache.

NOTE	The cache uses the search criteria as the key to cached items. Search requests with different criteria are cached as separate items. For example, suppose you send a search request specifying that you just want to retrieve the <code>uid</code> attribute. Your client caches the result of that search. If you send a similar search request specifying that you want to retrieve all attributes instead of just the <code>uid</code> , the result cached from the previous search are not used.

The cache uses a combination of the following information as the key to a cached item:

- The host name and port number of the LDAP server that you are searching.
- The DN to which you are currently authenticated.

From the search criteria, the base DN, scope, filter, attributes to be returned, and an indication of whether to return attribute types only or attribute types and values.

Related Functions

Table 16-21 lists functions related to the `LDAPMemCache` structure.

Table 16-21 Functions for `LDAPMemCache` Structure

Function	Description
<code>ldap_memcache_init()</code>	Creates an <code>LDAPMemCache</code> structure and initialize the caching mechanism.
<code>ldap_memcache_set()</code>	Associates an <code>LDAPMemCache</code> structure with a connection handle (an LDAP structure).
<code>ldap_memcache_get()</code>	Returns the <code>LDAPMemCache</code> structure that is associated with a connection handle (an LDAP structure).
<code>ldap_memcache_update()</code>	Forces the cache to remove expired items.
<code>ldap_memcache_flush()</code>	Removes entries from the cache.
<code>ldap_memcache_destroy()</code>	Frees the memory used by the cache and frees the <code>LDAPMemCache</code> structure.

LDAPMessage

`LDAPMessage` is a structure representing the results of an LDAP operation, i.e.: a chain of search results, an entry in the search results, or a search reference in the search results.

NOTE LDAPMessage is not completely defined in ldap.h because the fields within the structure are not intended to be directly accessible to clients.

Description

Calling the ldap_search_ext_s() or ldap_search_ext() followed by the ldap_result() function creates an LDAPMessage structure to represent the chain of results from the search. Calling the ldap_first_entry() or ldap_next_entry() function creates an LDAPMessage structure to represent an entry in the search results. Calling ldap_first_reference() or ldap_next_reference() creates an LDAPMessage structure to represent a search reference in the search results. To free the LDAPMessage structure, call the ldap_msgfree() routine.

LDAPMod

LDAPMod is a structure that you use to specify changes to an attribute in a directory entry.

Description

Before you call the ldap_add_ext(), ldap_add_ext_s(), ldap_modify_ext(), or ldap_modify_ext_s() functions to add or modify an entry in the directory, you need to fill LDAPMod structures with the attribute values that you intend to add or change. Code Example 16-25 illustrates the LDAPMod definition.

Code Example 16-25 LDAPMod Definition

```
typedef struct ldapmod {
    int mod_op;
    char *mod_type;
    union {
        char **modv_strvals;
        struct berval **modv_bvals;
    } mod_vals;
#define mod_values mod_vals.modv_strvals
#define mod_bvalues mod_vals.modv_bvals
} LDAPMod;
```

Table 16-22 defines the fields in this structure.

Table 16-22 LDAPMod Field Descriptions

Field	What It Contains
mod_op	The operation to be performed on the attribute and the type of data specified as the attribute values. This field can have one of the following values: - LDAP_MOD_ADD adds a value to the attribute. - LDAP_MOD_DELETE removes the value from the attribute. - LDAP_MOD_REPLACE replaces all existing values of the attribute. In addition, if you are specifying binary values in the mod_bvalues field, you should use the bitwise OR operator () to combine LDAP_MOD_BVALUES with the operation type. For example: <pre>mod->mod_op = LDAP_MOD_ADD LDAP_MOD_BVALUES</pre> Note the following caveats when defining : - If you specify LDAP_MOD_DELETE and remove all values in an attribute, the attribute is removed from the entry. - If you specify LDAP_MOD_DELETE and NULL in the mod_values field, the attribute is removed from the entry. - If you specify LDAP_MOD_REPLACE and NULL in the mod_values field, the attribute is removed from the entry. - If you specify LDAP_MOD_REPLACE and the attribute does not exist in the entry, the attribute is added to the entry. - If you specify LDAP_MOD_ADD and the attribute does not exist in the entry, the attribute is added to the entry. Also, if you are using the structure to add a new entry, you can specify 0 for the mod_op field (unless you are adding binary values and need to specify LDAP_MOD_BVALUES). See “Adding an Entry” on page 149 of Chapter 7, “Adding, Modifying and Deleting Entries” for details.
mod_type	The attribute type that you want to add, delete, or replace the values of (for example, sn or telephoneNumber).
mod_values	A pointer to a NULL-terminated array of string values for the attribute.
mod_bvalues	A pointer to a NULL-terminated array of berval structures for the attribute.

If you’ve allocated memory for the structures yourself, you should free them when finished by calling the `ldap_mods_free()` function.

Example

Code Example 16-26 sets up an LDAPMod structure to change the email address of a user’s entry to `bjensen@example.com`.

Code Example 16-26 Setting Up an LDAPMod Structure

```

LDAP *ld;
LDAPMod attribute1;
LDAPMod *list_of_attrs[2];
char *mail_values[] = { "bjensen@example.com", NULL };
char *dn;
...
/* Identify the entry that you want changed */
char *dn = "uid=bjensen,ou=People,dc=example,dc=com";

/* Specify that you want to replace the value of an attribute */
attribute1.mod_op = LDAP_MOD_REPLACE;

/* Specify that you want to change the value of the mail attribute */
attribute1.mod_type = "mail";

/* Specify the new value of the mail attribute */
attribute1.mod_values = mail_values;

/* Add the change to the list of attributes that you want changed */
list_of_attrs[0] = &attribute_change;
list_of_attrs[1] = NULL;

/* Update the entry with the change */
if ( ldap_modify_s( ld, dn, list_of_attrs ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_modify_s" );
    return( 1 );
}
...

```

LDAP_REALLOC_CALLBACK

This callback function prototype of `ldap_memalloc_fns` represents memory allocation.

See Also

`ldap_memalloc_fns`

LDAP_REBINDPROC_CALLBACK

`LDAP_REBINDPROC_CALLBACK` specifies the prototype for a callback function that retrieves authentication information when automatically following referrals to other servers.

Description

If you define a function with this prototype and specify it when calling the `ldap_set_rebind_proc()` function, the function will be called by your LDAP client. Code Example 16-27 is the prototype.

Code Example 16-27 LDAP_REBINDPROC_CALLBACK Prototype

```
typedef int (LDAP_CALL LDAP_CALLBACK
LDAP_REBINDPROC_CALLBACK)( LDAP *ld, char **dn, char **passwdp,
int *authmethodp, int freeit, void *arg);
```

See Also

`ldap_set_rebind_proc()`

LDAPsortkey

LDAPsortkey represents the criteria to be used by a server control to sort search results before sending the results back to the client.

NOTE

Controls are part of the LDAPv3 and are used to extend the functionality of an LDAP operation. The two types of controls described in the LDAPv3 are:

- *Server controls* are controls that are sent from the client to the server along with an LDAP request. (In some cases, a server can include a control in the response it sends back to the client.) For example, you can include a server control in a search request to specify that you want the server to sort the search results before sending them back. See “Using the Server-Side Sorting Control” on page 237 of Chapter 13, “Working with LDAP Controls” for more information.
- *Client controls* are controls that can extend the client but that are never sent to the server. As a general example, you might be able to pass a client control to an LDAP API function, which might parse the control and use the data that you’ve specified in the control. Sun Java System LDAP SDK for C does not currently support any client controls.

Description

Code Example 16-28 illustrates how LDAPsortkey is defined.

Code Example 16-28 LDAPSsortkey Definition

```
typedef struct LDAPSsortkey {
    char *sk_attrtype;
    char *sk_matchruleoid;
    int sk_reverseorder;
} LDAPSsortkey;
```

Table 16-23 defines the fields in this structure.

Table 16-23 LDAPSsortkey Field Descriptions

Field	What It Contains
sk_attrtype	Pointer to a NULL-terminated string that specifies the name of the attribute that you want to use for sorting.
sk_matchruleoid	Pointer to a NULL-terminated string that specifies the object identifier (OID) of the matching rule that you want to use for sorting.
sk_reverseorder	Specifies whether or not the results are sorted in reverse order. This field can have one of the following values: - A non-zero value specifies that the results should be sorted in reverse order. 0 specifies that the results should be sorted in normal (forward) order.

As an example, you can use LDAPSsortkey to specify that you want the server to sort search results by the roomNumber attribute. To create an array of LDAPSsortkey structures, you can call the ldap_create_sort_keylist() function. To free an array of LDAPSsortkey structures, you can call the ldap_free_sort_keylist() function.

LDAP_TF_GET_ERRNO_CALLBACK

LDAP_TF_GET_ERRNO_CALLBACK specifies the prototype for a callback function to get the value of the errno variable for a thread.

NOTE For more details on the errno variable, see ldap_thread_fns.

Description

Code Example 16-29 is the prototype specified by this data type.

Code Example 16-29 LDAP_TF_GET_ERRNO_CALLBACK Prototype

```
typedef int (LDAP_C LDAP_CALLBACK
LDAP_TF_GET_ERRNO_CALLBACK) ( void );
```

LDAP_TF_GET_LDERRNO_CALLBACK

LDAP_TF_GET_LDERRNO_CALLBACK specifies the prototype for a callback function to retrieve the LDAP result code for an operation.

Description

If you define a function with this prototype and set it in the `ldap_thread_fns` structure, your callback function is called when the `ldap_get_lderrno()` function is called. The arguments of `ldap_get_lderrno()` are passed to your function, and the value returned by your function is returned by `ldap_get_lderrno()`.

NOTE

For more details on the arguments and return values that your function must use, see `ldap_get_lderrno()`.

Code Example 16-30 is the prototype for this data type.

Code Example 16-30 LDAP_TF_GET_LDERRNO_CALLBACK Prototype

```
typedef int (LDAP_C LDAP_CALLBACK
LDAP_TF_GET_LDERRNO_CALLBACK)( char **, char **, void * );
```

LDAP_TF_SET_ERRNO_CALLBACK

LDAP_TF_SET_ERRNO_CALLBACK specifies the prototype for a callback function to set the value of the `errno` variable for a thread.

NOTE

For more details on the `errno` variable, see `ldap_thread_fns`.

Description

Code Example 16-31 is the prototype for this data type.

Code Example 16-31 LDAP_TF_SET_ERRNO_CALLBACK Prototype

```
typedef void (LDAP_C LDAP_CALLBACK
LDAP_TF_SET_ERRNO_CALLBACK)( int );
```

LDAP_TF_SET_LDERRNO_CALLBACK

LDAP_TF_SET_LDERRNO_CALLBACK specifies the prototype for a callback function to set the LDAP result code for an operation.

NOTE

For more details on the arguments that your function must use, see `ldap_set_lderrno()`.

Description

If you define a function with this prototype and set it in the `ldap_thread_fns` structure, your callback function is called when the `ldap_set_lderrno()` function is called. The arguments of `ldap_set_lderrno()` are passed to your function. Code Example 16-32 is the prototype specified by this data type.

Code Example 16-32 LDAP_TF_SET_LDERRNO_CALLBACK Prototype

```
typedef void (LDAP_C LDAP_CALLBACK
LDAP_TF_SET_LDERRNO_CALLBACK)( int, char *, char *, void * );
```

LDAP_TF_MUTEX_ALLOC_CALLBACK

LDAP_TF_MUTEX_ALLOC_CALLBACK specifies the prototype for a callback function to allocate a mutex.

Description

Code Example 16-33 is the prototype specified by this data type.

Code Example 16-33 LDAP_TF_MUTEX_ALLOC_CALLBACK Prototype

```
typedef void *(LDAP_C LDAP_CALLBACK
LDAP_TF_MUTEX_ALLOC_CALLBACK)( void );
```

See Also

ldap_thread_fns

LDAP_TF_MUTEX_FREE_CALLBACK

LDAP_TF_MUTEX_FREE_CALLBACK specifies the prototype for a callback function to free a mutex.

Description

Code Example 16-34 is the prototype specified by this data type.

Code Example 16-34 LDAP_TF_MUTEX_FREE_CALLBACK Prototype

```
typedef void (LDAP_C LDAP_CALLBACK
LDAP_TF_MUTEX_FREE_CALLBACK)( void * );
```

See Also

ldap_thread_fns

LDAP_TF_MUTEX_LOCK_CALLBACK

LDAP_TF_MUTEX_LOCK_CALLBACK specifies the prototype for a callback function to lock a mutex.

Description

Code Example 16-35 is the prototype specified by this data type.

Code Example 16-35 LDAP_TF_MUTEX_LOCK_CALLBACK Prototype

```
typedef int (LDAP_C LDAP_CALLBACK
LDAP_TF_MUTEX_LOCK_CALLBACK)( void * );
```

See Also

ldap_thread_fns

LDAP_TF_MUTEX_TRYLOCK_CALLBACK

This function prototype is not supported in this release of the LDAP SDK for C.

LDAP_TF_MUTEX_UNLOCK_CALLBACK

LDAP_TF_MUTEX_UNLOCK_CALLBACK specifies the prototype for a callback function to unlock a mutex.

Description

Code Example 16-36 is the prototype specified by this data type.

Code Example 16-36 LDAP_TF_MUTEX_UNLOCK_CALLBACK Prototype

```
typedef int (LDAP_C LDAP_CALLBACK
LDAP_TF_MUTEX_UNLOCK_CALLBACK)( void * );
```

See Also

ldap_thread_fns

LDAP_TF_SEMA_ALLOC_CALLBACK

This function prototype is not supported in this release of the LDAP SDK for C.

LDAP_TF_SEMA_FREE_CALLBACK

This function prototype is not supported in this release of the LDAP SDK for C.

LDAP_TF_SEMA_POST_CALLBACK

This function prototype is not supported in this release of the LDAP SDK for C.

LDAP_TF_SEMA_WAIT_CALLBACK

This function prototype is not supported in this release of the LDAP SDK for C.

LDAP_TF_THREADID_CALLBACK

LDAP_TF_THREADID_CALLBACK specifies the prototype for a callback function to return an identifier that is unique to the calling thread, much like the POSIX `pthread_self()` function does.

Description

Code Example 16-37 is the prototype specified by this data type.

Code Example 16-37 LDAP_TF_THREADID_CALLBACK Prototype

```
typedef void *(LDAP_C LDAP_CALLBACK
LDAP_TF_THREADID_CALLBACK)( void );
```

See Also

`ldap_thread_fns`

ldap_thread_fns

The `ldap_thread_fns` structure contains a set of pointers to functions that you want to use when write a multithreaded client.

Description

Code Example 16-38 is the definition of the `ldap_thread_fns` structure.

Code Example 16-38 ldap_thread_fns Structure Definition

```
struct ldap_thread_fns {
    LDAP_TF_MUTEX_ALLOC_CALLBACK *ltf_mutex_alloc;
    LDAP_TF_MUTEX_FREE_CALLBACK *ltf_mutex_free;
    LDAP_TF_MUTEX_LOCK_CALLBACK *ltf_mutex_lock;
    LDAP_TF_MUTEX_UNLOCK_CALLBACK *ltf_mutex_unlock;
    LDAP_TF_GET_ERRNO_CALLBACK *ltf_get_errno;
    LDAP_TF_SET_ERRNO_CALLBACK *ltf_set_errno;
    LDAP_TF_GET_LDERRNO_CALLBACK *ltf_get_lderrno;
    LDAP_TF_SET_LDERRNO_CALLBACK *ltf_set_lderrno;
    void *ltf_lderrno_arg;
};
```

Table 16-24 defines the fields in this structure.

Table 16-24 ldap_thread_fns Field Descriptions

Field	What It Contains
ltf_mutex_alloc	Function pointer for allocating a mutex. This function is called by the client when needed if the function pointer is not NULL. The function must have the prototype specified by LDAP_TF_MUTEX_ALLOC_CALLBACK.
ltf_mutex_free	Function pointer for freeing a mutex. This function is called by the client when needed if the function pointer is not NULL. The function must have the prototype specified by LDAP_TF_MUTEX_FREE_CALLBACK.
ltf_mutex_lock	Function pointer for locking critical sections of code. This function is called by the client when needed if the function pointer is not NULL. The function must have the prototype specified by LDAP_TF_MUTEX_LOCK_CALLBACK.
ltf_mutex_unlock	Function pointer for unlocking critical sections of code. This function is called by the client when needed if the function pointer is not NULL. The function must have the prototype specified by LDAP_TF_MUTEX_UNLOCK_CALLBACK.
ltf_get_errno	Function pointer for getting the value of the errno variable. This function is called by the client when needed if the function pointer is not NULL. In a threaded environment, errno is typically redefined at compile time so that it has a value for each thread, rather than a global value for the entire process. Because the libldap library does not know what method your code and threading environment will use to get the value of errno for each thread, it calls this function to obtain the value. The function must have the prototype specified by LDAP_TF_GET_ERRNO_CALLBACK.

Table 16-24 ldap_thread_fns Field Descriptions (*Continued*)

Field	What It Contains
ltf_set_errno	Function pointer for setting the value of the errno variable. This function is called by the client when needed if the function pointer is not NULL. In a threaded environment, errno is typically redefined at compile time so that it has a value for each thread, rather than a global value for the entire process. Because the libldap library does not know what method your code and threading environment will use to get the value of errno for each thread, it calls this function to obtain the value. The function must have the prototype specified by LDAP_TF_SET_ERRNO_CALLBACK.
ltf_get_lderrno	Function pointer for getting error values from calls to functions in the libldap library. This function is called by the client as needed if the function pointer isn't NULL. If this function pointer is not set, the libldap library records these errors in fields in the LDAP structure. The function must have the prototype specified by LDAP_TF_GET_LDERRNO_CALLBACK.
ltf_set_lderrno	Function pointer for setting error values from calls to functions in the libldap library. This function is called by the client as needed if the function pointer isn't NULL. If this function pointer is not set, the libldap library records these errors in fields in the LDAP structure. The function must have the prototype specified by LDAP_TF_SET_LDERRNO_CALLBACK.
ltf_lderrno_arg	Additional parameter passed to the functions for getting and setting error values from calls to functions in the libldap library. (*ltf_get_lderrno) and (*ltf_set_lderrno) identify these functions.

Example

For an example of setting up the ldap_thread_fns structure, see Chapter 14, “Writing Multithreaded Clients.”

LDAPURLDesc

LDAPURLDesc is a structure that represents the components of an LDAP URL.

Description

LDAP URLs have the following syntax:

```
ldap://host:port/dn?attributes?scope?(filter)
```

Where, for example:

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

Calling the ldap_url_parse() function creates an LDAPURLDesc structure with the components of the URL. Code Example 16-39 is the LDAPURLDesc definition.

Code Example 16-39 LDAPURLDesc Definition

```
typedef struct ldap_url_desc {
    char *lud_host;
    int lud_port;
    char *lud_dn;
    char **lud_attrs;
    int lud_scope;
    char *lud_filter;
    unsigned long lud_options;
} LDAPURLDesc;
```

Table 16-25 describes the fields in this structure.

Table 16-25 LDAPURLDesc Field Descriptions

Field	What It Contains
lud_host	Name of the host in the URL.
lud_port	Number of the port in the URL.
lud_dn	Distinguished name in the URL. This <i>base entry</i> DN identifies the starting point of the search.
lud_attrs	Pointer to a NULL-terminated list of the attributes specified in the URL.
lud_scope	Integer representing the scope of the search specified in the URL: - LDAP_SCOPE_BASE specifies a search of the base entry. - LDAP_SCOPE_ONELEVEL specifies a search of all entries one level under the base entry (not including the base entry). - LDAP_SCOPE_SUBTREE specified a search of all entries at all levels under the base entry (including the base entry).
lud_filter	Search filter included in the URL.
lud_options	Options (if LDAP_URL_OPT_SECURE, indicates that the protocol is ldaps:// instead of ldap://).

For example, suppose you pass the following URL to the ldap_url_parse() function:

```
ldap://ldap.ipplanet.com:5000/dc=example,dc=com?cn,mail, \
telephoneNumber?sub?(sn=Jensen)
```

The resulting LDAPURLDesc structure (ludpp, in this example) will contain the values in Table 16-26.

Table 16-26 LDAPURLDesc Parsed URL Values

Component	Value
ludpp->lud_host	ldap.ipplanet.com
ludpp->lud_port	5000
ludpp->lud_dn	dc=example,dc=com
ludpp->lud_attrs[0]	cn
ludpp->lud_attrs[1]	mail
ludpp->lud_attrs[2]	telephoneNumber
ludpp->lud_attrs[3]	NULL
ludpp->lud_scope	LDAP_SCOPE_SUBTREE
ludpp->lud_filter	(sn=Jensen)

To free the LDAPURLDesc structure, call the `ldap_free_urldesc()` routine.

LDAP_VALCMP_CALLBACK

LDAP_VALCMP_CALLBACK specifies the prototype for a callback function to sort a specified set of values.

Description

If you define a function with this prototype and specify it when calling the `ldap_sort_strcasecmp()` function, the function will be called by your LDAP client. Code Example 16-40 is the prototype.

Code Example 16-40 LDAP_VALCMP_CALLBACK Prototype

```
typedef int (LDAP_C LDAP_CALLBACK
LDAP_VALCMP_CALLBACK)(const char**, const char**);
```

ldap_sort_values() requires a function pointer that matches the signature defined by this type. ldap_sort_strcasecmp() is a value comparison function defined using this type which can therefore be passed to ldap_sort_values().

See Also

ldap_sort_values(), ldap_sort_strcasecmp()

LDAPVersion

The LDAPVersion structure contains version information about the LDAP SDK for C.

CAUTION The LDAPVersion type is deprecated; it is documented here only for backward compatibility. Instead, use the ldap_get_option() function with the LDAP_OPT_API_INFO option and retrieve an LDAPAPIInfo structure.

Description

Call the ldap_version() function to return a pointer to an LDAPVersion structure containing version information. Code Example 16-41 is the definition for LDAPVersion.

Code Example 16-41 LDAPVersion Definition

```
typedef struct _LDAPVersion {
    int sdk_version;
    int protocol_version;
    int SSL_version;
    int security_level;
} LDAPVersion;
```

Table 16-27 describes the fields in this structure.

Table 16-27 LDAPVersion Field Definitions

Field	What It Contains
sdk_version	Version number of the LDAP SDK for C multiplied by 100 (for example, the value 100 in this field represents version 1.0).
protocol_version	Highest supported LDAP protocol version multiplied by 100 (for example, the value 300 in this field represents LDAPv3).

Table 16-27 LDAPVersion Field Definitions (*Continued*)

Field	What It Contains
SSL_version	Supported SSL version multiplied by 100 (for example, the value 300 in this field represents SSL 3.0).
security_level	Level of encryption supported in bits (for example, 128 for domestic or 40 for export). If SSL is not enabled, the value of this field is LDAP_SECURITY_NONE.

LDAPVirtualList

LDAPVirtualList specifies the information that can be used to create a *virtual list view* control.

Description

A *virtual list view* control is a LDAPv3 control designed to allow the client to retrieve subsets of search results to display in a *virtual list box*. A *virtual list box* is typically a graphical user interface that displays a long list of entries with a few entries visible. To display the list, the client usually does not retrieve the entire list of entries from the server. Instead, the client just retrieves the subset of entries to be displayed. (For more information, see “Using the Virtual List View Control” on page 245 of Chapter 13, “Working with LDAP Controls.”)

After you create an LDAPVirtualList structure and specify values for its fields, you can create the virtual list view control by calling the function `ldap_create_virtuallist_control()`. You can pass this control and a server-side sorting control (created by calling the `ldap_create_sort_keylist()` function and the `ldap_create_sort_control()` function) to the `ldap_search_ext()` or `ldap_search_ext_s()` function. To get the virtual list view response control sent back from the server, call the `ldap_parse_result()` function to get the list of controls returned by the server, then call the `ldap_parse_virtuallist_control()` function to retrieve information from the control. End users can display different sections of the list by scrolling up or down.

NOTE

The object identifier (OID) for this control is OID 2.16.840.1.113730.3.4.9, or LDAP_CONTROL_VLVREQUEST as defined in `ldap.h`. It is supported by Sun Java System Directory Server. For information on determining if your server supports this or other LDAPv3 controls, see “Determining Supported Controls” on page 234 of Chapter 13, “Working with LDAP Controls.”

The virtual list view control provides the means for your client to request and retrieve certain subsets of a long, sorted list of entries. The control specifies the following information:

- The entry in the list that is currently selected.
- The number of entries to be displayed in the list before the selected item.
- The number of entries to be displayed in the list after the selected entry.

The currently selected entry can be identified in one of the following ways:

- By the index of the entry in the entire list (in which case, the control specifies both the offset of the entry and the total number of entries in the list).
- By the value of the entry (in which case, the control specifies that value).

For example, a virtual list view control might specify that you want to retrieve entries 15 through 24 in a list of 100 results with entry 20 being the selected entry. The control uses the following information to specify this:

- The selected entry is the 20th entry from the top (in other words, the index or offset of the entry is 20) of a list of 100.
- Get 5 entries before the selected entry in the list (entries 15 - 19).
- Get 4 entries after the selected entry in the list (entries 21 - 24).

As another example, a virtual list view control might specify that you want to retrieve a subset of entries that start with the letter "c" or a later letter in the alphabet. The control might specify the following information:

- The selected entry is the first entry that starts with the letter "c". (The size of the list is not relevant in determining the selected entry in this case.)
- Get 5 entries before the selected entry in the list.
- Get 4 entries after the selected entry in the list.

Code Example 16-42 illustrates the LDAPVirtualList definition.

Code Example 16-42 LDAPVirtualList Definition

```
typedef struct ldapvirtuallist {
    unsigned long ldvlist_before_count;
    unsigned long ldvlist_after_count;
    char *ldvlist_attrvalue;
    unsigned long ldvlist_index;
    unsigned long ldvlist_size;
```


Code Example 16-42 LDAPVirtualList Definition (*Continued*)

```
void *ldvlist_extradata;
} LDAPVirtualList;
```

Table 16-28 describes the fields in this structure.

Table 16-28 LDAPVirtualList Field Descriptions

Field	What It Contains
ldvlist_before_count	Number of entries before the selected entry that you want to retrieve.
ldvlist_after_count	Number of entries after the selected entry that you want to retrieve.
ldvlist_attrvalue	Specifies the value that you want to find in the list. The selected entry in the list is the first entry that is greater than or equal to this value. If this field is NULL, the ldvlist_index and ldvlist_size fields are used to determine the selected entry instead.
ldvlist_index	If the ldvlist_attrvalue field is NULL, specifies the offset or index of the selected entry in the list. This field is used in conjunction with the ldvlist_size field to identify the selected entry.
ldvlist_size	If the ldvlist_attrvalue field is NULL, specifies the total number items in the list. This field is used in conjunction with the ldvlist_index field to identify the selected entry.
ldvlist_extradata	Reserved for application-specific use. This data is not used in the virtual list view control.

LDAP_X_EXTIOF_CLOSE_CALLBACK

This specifies the prototype for an extended I/O callback function to close a socket connection.

Description

Code Example 16-43 is the prototype.

Code Example 16-43 LDAP_X_EXTIOF_CLOSE_CALLBACK Prototype

```
typedef int (LDAP_C LDAP_CALLBACK LDAP_X_EXTIOF_CLOSE_CALLBACK )(
    int s, void *arg );
```

See Also

ldap_x_ext_io_fns

LDAP_X_EXTIOF_CONNECT_CALLBACK

This specifies the prototype for an extended I/O callback function to open a socket connection.

Description

LDAP_X_EXTIOF_CONNECT_CALLBACK takes a timeout parameter whose value (in milliseconds) can be set by calling ldap_set_option(). Code Example 16-44 is the prototype.

Code Example 16-44 LDAP_X_EXTIOF_CONNECT_CALLBACK Prototype

```
typedef int (LDAP_C LDAP_CALLBACK LDAP_X_EXTIOF_CONNECT_CALLBACK )(
    int s, LDAP_X_SockAddr *saddr, int saddr_len,
    int timeout /* milliseconds */, void *arg );
```

See Also

ldap_x_ext_io_fns

LDAP_X_EXTIOF_DISPOSEHANDLE_CALLBACK

This specifies the prototype for an extended I/O callback function to dispose of a session handle.

Description

LDAP_X_EXTIOF_DISPOSEHANDLE_CALLBACK is called right before the LDAP session handle and its contents are destroyed (during ldap_unbind()). The value of the arg parameter passed to the callback is always the default one. Code Example 16-45 is the prototype.

Code Example 16-45 LDAP_X_EXTIOF_DISPOSEHANDLE_CALLBACK Prototype

```
typedef void (LDAP_C LDAP_CALLBACK LDAP_X_EXTIOF_DISPOSEHANDLE_CALLBACK)(
    LDAP *ld, void *arg );
```

See Also

ldap_x_ext_io_fns

LDAP_X_EXTIOF_NEWHANDLE_CALLBACK

This specifies the prototype for an extended I/O callback function to define a new session handle.

Description

LDAP_X_EXTIOF_NEWHANDLE_CALLBACK is called right after the LDAP session handle is created (during `ldap_init()`). The value of the `arg` parameter passed to the callback is always the default one. If the callback returns anything other than `LDAP_SUCCESS`, the session handle allocation fails. Code Example 16-46 is the prototype.

Code Example 16-46 LDAP_X_EXTIOF_NEWHANDLE_CALLBACK Prototype

```
typedef int (LDAP_C LDAP_CALLBACK LDAP_X_EXTIOF_NEWHANDLE_CALLBACK)(
    LDAP *ld, void *arg );
```

See Also

ldap_x_ext_io_fns

ldap_x_ext_io_fns

This is a structure to hold extended I/O function pointers. Code Example 16-47 is the definition.

Code Example 16-47 ldap_x_ext_io_fns Definition

```
struct ldap_x_ext_io_fns {
    /* lextiof_size should always be set to LDAP_X_EXTIOF_SIZE */
    int lextiof_size;
    LDAP_X_EXTIOF_CONNECT_CALLBACK *lextiof_connect;
    LDAP_X_EXTIOF_CLOSE_CALLBACK *lextiof_close;
    LDAP_X_EXTIOF_READ_CALLBACK *lextiof_read;
    LDAP_X_EXTIOF_WRITE_CALLBACK *lextiof_write;
    LDAP_X_EXTIOF_POLL_CALLBACK *lextiof_poll;
    LDAP_X_EXTIOF_NEWHANDLE_CALLBACK *lextiof_newhandle;
    LDAP_X_EXTIOF_DISPOSEHANDLE_CALLBACK *lextiof_disposehandle;
    void *lextiof_session_arg;
```

Code Example 16-47 ldap_x_ext_io_fns Definition

```
LDAP_X_EXTIOF_WRITEV_CALLBACK      *lxtiof_writev;
};
#define LDAP_X_EXTIO_FNS_SIZE      sizeof(struct ldap_x_ext_io_fns)
```

LDAP_X_EXTIOF_POLL_CALLBACK

This specifies the prototype for an extended I/O callback function to close a socket connection.

Description

The LDAP_X_EXTIOF_POLL_CALLBACK interface is provided instead of a standard select() interface. The timeout is in milliseconds. The value of the arg parameter passed to the callback is always the default one. Code Example 16-48 is the prototype.

Code Example 16-48 LDAP_X_EXTIOF_POLL_CALLBACK Prototype

```
typedef int (LDAP_C LDAP_CALLBACK LDAP_X_EXTIOF_POLL_CALLBACK)(
    LDAP_X_PollFD fds[], int nfds, int timeout /* milliseconds */,
    void *arg );
```

See Also

ldap_x_ext_io_fns, LDAP_X_PollFD

ldap_x_hostlist_status

This structure contains utility functions for parsing space-separated host lists.

Description

Code Example 16-49 is the definition of this structure.

Code Example 16-49 ldap_x_hostlist_status Definition

```
struct ldap_x_hostlist_status;
LDAP_API(int) LDAP_CALL ldap_x_hostlist_first( const char *hostlist,
    int defport, char **hostp, int *portp /* host byte order */,
    struct ldap_x_hostlist_status **statusp );
```

Code Example 16-49 ldap_x_hostlist_status Definition (*Continued*)

```
LDAP_API(int) LDAP_CALL ldap_x_hostlist_next( char **hostp,
                                             int *portp /* host byte order */, struct ldap_x_hostlist_status
                                             *status
);
LDAP_API(void) LDAP_CALL ldap_x_hostlist_statusfree(
    struct ldap_x_hostlist_status *status );
```

LDAP_X_PollFD

This is a structure to hold the set of file descriptors that will be surveyed for I/O by the LDAP_X_EXTIOF_POLL_CALLBACK.

Description

Code Example 16-50 is the definition of the structure.

Code Example 16-50 LDAP_X_PollFD Structure Definition

```
typedef struct ldap_x_pollfd { /* used by LDAP_X_EXTIOF_POLL_CALLBACK */
    int lpoll_fd; /* integer file descriptor / socket */
    void *lpoll_arg; /* pointer socket and for use by application */
    short lpoll_events; /* requested event */
    short lpoll_revents; /* returned event */
} LDAP_X_PollFD;
```

LDAP_X_PollFD

Function Reference

This chapter contains detailed reference material for each public function of the Sun™ Java System Directory Server Resource Kit LDAP SDK for C. Each reference gives a description of the function, and its usage syntax, parameters and return values. In many cases, an example program is also included. The chapter contains the following sections:

- Typographic Conventions
- Function Summary by Task
- Functions Alphabetically

Typographic Conventions

The general purpose of a function is usually given by its name. Table 17-1 lists the naming conventions used when adding suffixes to function names.

Table 17-1 Function Naming Conventions

Suffix	Description
_ext	Identifies extended functions introduced for LDAPv3. These functions are augmented through additional parameters, as compared to the functions they replace. As LDAPv3 has become widely adopted, it is recommended you use extended functions whenever possible in your new applications. See Table 17-19 on page 340.

Table 17-1 Function Naming Conventions (*Continued*)

Suffix	Description
_s	Identifies the synchronous form of functions. A synchronous function will not return until it receives a response from the server, thereby blocking the caller. Similar function names without this suffix are asynchronous, allowing the caller to perform other operations while waiting for a result. (The synchronous and asynchronous forms exist only for functions which involve possible delays due to communication with the LDAP server. Functions which do not involve communication do not use this suffix.) For more information on these options, see “Managing Memory” on page 86 of Chapter 4, “Using the LDAP API.”

Function Summary by Task

In the following sections, the functions of the LDAP SDK for C are grouped into task categories.

Working with Basic Encoding Rules

The functions listed in Table 17-2 may be used to encode and decode with Basic Encoding Rules (BER). They are often used inside of control and extension values.

Table 17-2 Functions to Encode and Decode BER

Function	Description
ber_alloc() ber_alloc_t() ber_sockbuf_alloc() ber_special_alloc()	Allocate new BER elements.
ber_bvecfree() ber_bvfree() ber_free() ber_sockbuf_free() ber_sockbuf_free_data()	Free allocated memory.
ber_bvdup() ber_dup()	Duplicate BER structures.
ber_init() ber_init_w_nullchar()	Construct new structures.

Table 17-2 Functions to Encode and Decode BER (*Continued*)

Function	Description
ber_get_boolean() ber_get_int() ber_get_next() ber_get_next_buffer() ber_get_next_buffer_ext() ber_get_null() ber_get_option() ber_get_stringa() ber_get_stringal() ber_get_stringb()	Retrieve miscellaneous information.
ber_put_bitstring() ber_put_boolean() ber_put_enum() ber_put_int() ber_put_null() ber_put_ostring() ber_put_seq() ber_put_set() ber_put_string()	Write miscellaneous information.
ber_first_element() ber_next_element() ber_peek_tag() ber_skip_tag()	Target specific elements and tags.
ber_set_option() ber_set_string_translators()	Set session parameters.
ber_flatten() ber_printf() ber_read() ber_reset() ber_scanf() ber_sockbuf_get_option() ber_sockbuf_set_option() ber_start_seq() ber_start_seqq() ber_start_set() ber_svecfree() ber_write()	Miscellaneous functions.

Managing LDAP Sessions

The functions listed in Table 17-3 manage the different phases of an LDAP session: initialization, configuration, authentication (binding) and termination (unbinding). These phases are described in more detail in Chapter 3, “Writing an LDAP Client.”

Table 17-3 Functions to Manage an LDAP Session

Function	Description
ldap_init()	Initializes an LDAP session.
ldapsl_init()	Initializes an LDAP session over SSL.
ldapsl_pkcs_init()	Initializes a thread-safe session over SSL.
ldap_get_option() ldap_set_option()	Read and writes the current preference settings for a session.
ldap_memcache_init() ldap_memcache_set() ldap_memcache_get() ldap_memcache_update() ldap_memcache_flush() ldap_memcache_destroy()	Creates a memory cache and uses it for search results from this session. Manages the data in the cache and frees the memory when the cache is no longer needed. The in-memory cache for search results is a Sun Java System extension to the API.
ldap_unbind_ext()	Ends an LDAP session and frees the associated data structures.

Performing LDAP Operations

The functions listed in Table 17-4 perform LDAP operations on a server and retrieve the results. These operations are covered in Chapter 7, “Adding, Modifying and Deleting Entries,” Chapter 8, “Comparing Entries,” and Chapter 15, “Extended Operations.”

Table 17-4 Functions to Perform Operations on LDAP Server

Function	Description
ldap_add_ext() or ldap_add_ext_s()	Adds a new entry to the directory.
ldap_modify_ext() or ldap_modify_ext_s()	Modifies an entry in the directory
ldap_delete_ext() or ldap_delete_ext_s()	Deletes an entry from the directory.
ldap_rename() or ldap_rename_s()	Renames or moves one or more entry(ies).

Table 17-4 Functions to Perform Operations on LDAP Server (*Continued*)

Function	Description
<code>ldap_search_ext()</code> or <code>ldap_search_ext_s()</code>	Searches the directory.
<code>ldap_compare_ext()</code> or <code>ldap_compare_ext_s()</code>	Compares a value with the values of an entry's attribute.
<code>ldap_result()</code>	Checks the results of an asynchronous operation.
<code>ldap_abandon_ext()</code>	Cancels an asynchronous operation.
<code>ldap_extended_operation()</code> or <code>ldap_extended_operation_s()</code>	Performs an LDAPv3 extended operation.
<code>ldap_parse_extended_result()</code>	Parses the results of an LDAPv3 extended operation.
<code>ldap_msgfree()</code>	Frees the memory used by the data structure for the results.
<code>ldap_simple_bind()</code> or <code>ldap_simple_bind_s()</code>	Authenticates to an LDAP server using a password.
<code>ldap_sasl_bind()</code> and <code>ldap_parse_sasl_bind_result()</code> or <code>ldap_sasl_bind_s()</code>	Authenticates to an LDAP server using a SASL mechanism.
<code>ldap_set_rebind_proc()</code>	Specifies the function used to get authentication information when following referrals.

Processing Search Results

Search results are given as a chain of `LDAPMessage` structures which can contain both entry messages and search reference messages. The functions listed in Table 17-5 retrieve the results of a search operation and process the data structures which are returned. These functions are introduced in “Getting Search Results” on page 114 of Chapter 5, “Searching a LDAP Directory.”

Table 17-5 Functions to Process Search Results

Function	Description
<code>ldap_first_message()</code>	Gets the first message (an entry or search reference) in a chain of search results.
<code>ldap_next_message()</code>	Gets the next message (an entry or search reference) in a chain of search results.
<code>ldap_count_messages()</code>	Counts the number of messages (entries and search references) in a chain of search results.

Table 17-5 Functions to Process Search Results (*Continued*)

Function	Description
<code>ldap_msgid()</code>	Gets the ID number of a message structure containing part of a result.
<code>ldap_msgtype()</code>	Determines whether a message structure contains an entry or a search reference.
<code>ldap_msgfree()</code>	Frees the memory allocated for search results or other LDAP operation results.
<code>ldap_first_entry()</code>	Gets the first entry in a chain of search results.
<code>ldap_next_entry()</code>	Gets the next entry in a chain of search results.
<code>ldap_count_entries()</code>	Counts the number of entries in a chain of search results.
<code>ldap_first_reference()</code>	Gets the first search reference in a chain of search results.
<code>ldap_next_reference()</code>	Gets the next search reference in a chain of search results.
<code>ldap_count_references()</code>	Counts the number of search references in a chain of search results.
<code>ldap_parse_reference()</code>	Extracts the referral strings from a search reference.

Reading the Contents of an Entry

The functions listed in Table 17-6 are used to access the contents of an entry returned as a search result. These functions are described in “Getting Distinguished Names for Each Entry” on page 125 and “Getting Attribute Names from an Entry” on page 127 of Chapter 5, “Searching a LDAP Directory.”

Table 17-6 Functions to Access Contents of an Entry

Function	Description
<code>ldap_get_dn()</code>	Get the DN for an entry.
<code>ldap_explode_dn()</code> <code>ldap_explode_rdn()</code>	Split a DN or relative DN into its components.
<code>ldap_explode_dns()</code>	Takes a DNS-style DN and breaks it up into its components.
<code>ldap_is_dns_dn()</code>	Determines the style of the DN.
<code>ldap_dn2ufn()</code>	Removes cryptic type names to make a DN easier to read.
<code>ldap_first_attribute()</code>	Get the name of the first attribute in an entry.
<code>ldap_next_attribute()</code>	Get the name of the next attribute in an entry.
<code>ldap_get_values()</code>	Get the string values of an attribute.

Table 17-6 Functions to Access Contents of an Entry (*Continued*)

Function	Description
<code>ldap_get_values_len()</code>	Get the binary values of an attribute.
<code>ldap_count_values()</code>	Count the string values of an attribute.
<code>ldap_count_values_len()</code>	Count the binary values of an attribute.
<code>ldap_value_free()</code>	Free the memory allocated for the string values of an attribute.
<code>ldap_value_free_len()</code>	Free the memory allocated for the binary values of an attribute.

Sorting Search Results

With the functions listed in Table 17-7, your LDAP client can sort the data structures returned as search results. These functions are demonstrated in “Sorting Entries by an Attribute” on page 132 and “Sorting the Values of an Attribute” on page 133 in Chapter 5, “Searching a LDAP Directory.”

Table 17-7 Client-side Sorting Functions

Function	Description
<code>ldap_sort_entries()</code>	Sorts search results by DN or by a single attribute.
<code>ldap_multisort_entries()</code>	Sorts search results by by multiple attributes
<code>ldap_keysort_entries()</code>	Sorts search results using key(s).
<code>ldap_sort_values()</code>	Sorts the values of an attribute.
<code>ldap_sort_strcasecmp()</code>	A case-insensitive comparison function that you can pass to <code>ldap_sort_values()</code> .

Sun Java System Directory Server and others also provide server-side sorting through LDAPv3 controls. The controls mechanism is described in Chapter 13, “Working with LDAP Controls,” including the following server-side sorting functions which are shown in “Using the Server-Side Sorting Control” on page 237.

Table 17-8 Server-side Sorting Functions

Function	Description
<code>ldap_create_sort_keylist()</code>	Define the criteria for sorting entries by attribute values.
<code>ldap_free_sort_keylist()</code>	Free the memory for the sort criteria data structure.

Table 17-8 Server-side Sorting Functions (*Continued*)

Function	Description
ldap_create_sort_control()	Create the sort control structure needed for server-side sorting.
ldap_control_free()	Free the memory for the sort control data structure.
ldap_parse_sort_control()	Analyze the server's response to the sort control.
ldap_controls_free()	Free the memory for the sort control response data structure.

Working with Search Filters

The functions listed in Table 17-9 retrieve and build filters using a filter configuration file. Use this mechanism to create complex filters to pass to the search functions. Filters and filter configuration files are described in Chapter 6, “Using Filter Configuration Files.”

Table 17-9 Functions to Initialize, Retrieve, and Build Filters

Function	Description
ldap_init_getfilter()	Read a filter configuration file into memory.
ldap_init_getfilter_buf()	Read a filter configuration string from a buffer.
ldap_getfilter_free()	Free the filter configuration data structure from memory.
ldap_set_filter_additions()	Specify the prefix and suffix to be added to all filters retrieved from the filter configuration.
ldap_getfirstfilter()	Retrieve the first matching filter from the filter configuration.
ldap_getnextfilter()	Retrieve the next matching filter from the filter configuration.
ldap_create_filter()	Build a filter without using the filter configuration file mechanism.

Using LDAPv3 Controls

Sun Java System Directory Server and others also provide additional functionality through LDAPv3 controls. The functions listed in Table 17-10 configure LDAPv3 control structures which ask the server to perform advanced operations. This mechanism is described in Chapter 13, “Working with LDAP Controls.”

Table 17-10 Functions to Manage LDAPv3 Controls

Function	Description
<code>ldap_create_persistentsearch_control()</code> <code>ldap_get_entry_controls()</code> <code>ldap_parse_entrychange_control()</code>	Track changes to an entry or a set of entries by setting the persistent search control and analyzing the entry change control sent by the server.
<code>ldap_create_sort_keylist()</code> <code>ldap_free_sort_keylist()</code> <code>ldap_create_sort_control()</code> <code>ldap_parse_sort_control()</code>	Define and send criteria for server-side sorting of search results and check the server's response
<code>ldap_create_virtuallist_control()</code> <code>ldap_parse_virtuallist_control()</code>	Use in conjunction with server-side sorting to limit and handle the number of results returned to the client.
<code>ldap_create_proxiedauth_control()</code>	Define a proxy authorization control to perform an operation under a different bind DN
<code>ldap_parse_result()</code>	Parse an LDAP response from the server to extract the result and the response to the control.

Using Sun Java System Extensions

The functions listed in Table 17-11 are utility functions for parsing space-separated host lists. This is useful for implementing an extended I/O CONNECT callback function.

Table 17-11 Sun Java System Extensions

Function	Description
<code>ldap_x_hostlist_first()</code> <code>ldap_x_hostlist_next()</code> <code>ldap_x_hostlist_status()</code>	Utility functions for parsing space-separated host lists.

Working with LDAP URLs

The functions listed in Table 17-12 interpret LDAP URLs (universal resource locators of the form `ldap://...`). For additional information, see Chapter 9, “Working with LDAP URLs.”

Table 17-12 Functions to Interpret LDAP URLs

Function	Description
<code>ldap_is_ldap_url()</code>	Determine if a URL is an LDAP URL.
<code>ldap_url_parse()</code>	Split up an LDAP URL into its components.
<code>ldap_url_search()</code> <code>ldap_url_search_s()</code> <code>ldap_url_search_st()</code>	Perform the search specified by an LDAP URL.
<code>ldap_free_urldesc()</code>	Free the memory allocated for a parsed URL.

Working with UTF-8 Encoding

The functions listed in Table 17-13 work with strings and characters that use UTF-8 encoding.

Table 17-13 Functions to Handle UTF-8 Encoding

Function	Description
<code>ldap_utf8len()</code>	Function to determine bytel length.
<code>ldap_utf8next()</code>	Function to get next character.
<code>ldap_utf8prev()</code>	Function to get previous character.
<code>ldap_utf8copy()</code>	Function to copy a character to a destination.
<code>ldap_utf8characters()</code>	Function to return the number of characters.
<code>ldap_utf8getcc()</code>	Function moves the pointer to the next character following a UCS-4 character.
<code>ldap_utf8strtok_r()</code>	Function moves the pointer to the next character following a UCS-4 character.
<code>ldap_utf8isalnum()</code> <code>ldap_utf8isalpha()</code> <code>ldap_utf8isdigit()</code> <code>ldap_utf8len()</code> <code>ldap_utf8isspace()</code>	Functions to determine characteristics of the character.

Handling Errors

The functions listed in Table 17-14 handle errors returned by the functions. Chapter 18, “Result Codes” lists all of the error and status codes used in the LDAP SDK for C. You can also find details about possible errors in the reference section for each function.

Table 17-14 Functions for Error Handling

Function	Description
<code>ldap_parse_result()</code>	Get the error code resulting from an asynchronous LDAP operation.
<code>ldap_get_lderrno()</code>	Get information about the last error that occurred.
<code>ldap_set_lderrno()</code>	Set information about an error.
<code>ldap_err2string()</code>	Get the error message for a specific error code.
<code>ldapssl_err2string()</code>	Get the error message for a specific SSL error code.

Managing Memory

The functions listed in Table 17-15 free memory allocated by the LDAP SDK for C functions. See Table 4-1 on page 86 in the “Managing Memory” section of Chapter 4, “Using the LDAP API” for a list of all functions which allocate memory that needs to be freed.

Table 17-15 Functions to Free Memory

Function	Description
<code>ber_bvfree()</code>	Free the memory allocated for a berval structure.
<code>ber_free()</code>	Free the memory of a BerElement structure.
<code>ldap_control_free()</code>	Free the memory of an LDAPControl structure.
<code>ldap_controls_free()</code>	Free an array of LDAPControl structures.
<code>ldap_free_friendlymap()</code>	Free the memory of a FriendlyMap structure
<code>ldap_free_sort_keylist()</code>	Free an array of LDAPsortkey structures
<code>ldap_free_urldesc()</code>	Free the memory of an LDAPURLDesc structure
<code>ldap_getfilter_free()</code>	Free the memory of an LDAPfiltDesc structure.
<code>ldap_memfree()</code>	General function for freeing memory of any other data structure
<code>ldap_mods_free()</code>	Free an array of LDAPMod structure.

Table 17-15 Functions to Free Memory (*Continued*)

Function	Description
<code>ldap_msgfree()</code>	Free the memory of an LDAPMessage structure.
<code>ldap_value_free()</code>	Free the memory allocated for the string values of an attribute.
<code>ldap_value_free_len()</code>	Free the memory allocated for the binary values of an attribute (an array of berval structures)

You may also manage memory directly with the functions in Table 17-16.

Table 17-16 Functions to Directly Manage Memory

Function	Description
<code>ldap_x_malloc()</code>	Allocates memory.
<code>ldap_x_calloc()</code>	Allocates memory for an array of elements.
<code>ldap_x_realloc()</code>	Changes the size of a memory block.
<code>ldap_x_free()</code>	Frees a block of memory.

Working with Netscape Portable Runtime

The functions listed in Table 17-17 tie libldap into Netscape Portable Runtime (NSPR).

Table 17-17 Functions to Work with Netscape Portable Runtime

Function	Description
<code>prldap_get_default_socket_info()</code>	Gets default socket information.
<code>prldap_get_session_info()</code>	Gets application-specific data.
<code>prldap_get_session_option()</code>	Gets a session option specific to the prldap layer.
<code>prldap_get_socket_info()</code>	Gets socket information.
<code>prldap_init()</code>	Creates a new NSPR session handle.
<code>prldap_install_routines()</code>	Installs NSPR functions for use with LDAP handle.
<code>prldap_set_default_socket_info()</code>	Sets default socket information.
<code>prldap_set_session_info()</code>	Sets application-specific data.
<code>prldap_set_session_option()</code>	Sets a session option specific to the prldap layer.

Table 17-17 Functions to Work with Netscape Portable Runtime (*Continued*)

Function	Description
<code>prldap_set_socket_info()</code>	Sets socket information.

Deprecated and Outdated Functions

The functions listed in first column of Table 17-18 should no longer be used; they have been superseded by the newer functions listed in the second column of Table 17-18. Deprecated functions are those being dropped from the standard or from the Sun Java System extensions to the LDAP SDK for C. Outdated functions are those which are not officially deprecated but which have newer, extended functions for performing the same operation.

NOTE Deprecated functions are not guaranteed to be implemented in future versions of the API. However, all of these functions are still implemented in this version for backwards compatibility. For maintenance reasons, they have often been reimplemented to simply call the newer function which performs the same task.

Table 17-18 Deprecated Functions and Their Replacement

Deprecated Function	Replacement
<code>ldap_ber_free()</code>	Replaced by <code>ber_free()</code> .
<code>ldap_bind()</code> <code>ldap_bind_s()</code>	Replaced by <code>ldap_simple_bind()</code> and <code>ldap_simple_bind_s()</code> , respectively.
<code>ldap_build_filter()</code>	Replaced by <code>ldap_create_filter()</code> .
<code>ldap_cache_flush()</code>	Replaced by <code>ldap_memcache_*</code> () functions.
<code>ldap_chararray_add()</code> <code>ldap_chararray_dup()</code> <code>ldap_chararray_free()</code> <code>ldap_chararray_inlist()</code> <code>ldap_chararray_merge()</code> <code>ldap_chararray_position()</code> <code>ldap_str2chararray()</code>	None documented to replace these character array functions.
<code>ldap_create_proxyauth_control()</code>	Replaced by Description.
<code>ldap_get_lang_values()</code> <code>ldap_get_lang_values_len()</code>	Rarely used, no replacement.
<code>ldap_modrdn()</code>	Replaced by <code>ldap_modrdn2()</code> , itself outdated by <code>ldap_rename()</code> .

Table 17-18 Deprecated Functions and Their Replacement (*Continued*)

Deprecated Function	Replacement
ldap_modrdn_s()	Replaced by ldap_modrdn2_s(), itself outdated by ldap_rename_s().
ldap_open()	Replaced by ldap_init() and one of the bind functions.
ldap_result2error()	Replaced by ldap_parse_result().
ldap_setfilteraffixes()	Replaced by ldap_set_filter_additions().
ldap_ufn_search_c() ldap_ufn_search_ct() ldap_ufn_search_s() ldap_ufn_setfilter() ldap_ufn_setprefix() ldap_ufn_timeout()	None documented to replace these user-friendly search functions.
ldap_version()	
	Replaced by ldap_get_option().

Outdated Standard Functions and Equivalents

Many LDAPv2 functions are outdated now that LDAPv3 has been widely adopted. Table 17-19 lists both the outdated function and its new equivalent. Although LDAPv2 functions remain in the standard specification, you should use the LDAPv3 functions in any new client applications. Again, for maintenance reaons, the outdated functions have often been reimplemented to simply call the newer function.

Table 17-19 Outdated LDAPv2 Functions and New Equivalent

LDAPv2 Functions	LDAPv3 Equivalent
ldap_abandon()	ldap_abandon_ext()
ldap_add() ldap_add_s()	ldap_add_ext() ldap_add_ext_s()
ldap_compare() ldap_compare_s()	ldap_compare_ext() ldap_compare_ext_s()
ldap_delete() ldap_delete_s()	ldap_delete_ext() ldap_delete_ext_s()
ldap_modify() ldap_modify_s()	ldap_modify_ext() ldap_modify_ext_s()
ldap_modrdn2() ldap_modrdn2_s()	ldap_rename() ldap_rename_s()

Table 17-19 Outdated LDAPv2 Functions and New Equivalent (*Continued*)

LDAPv2 Functions	LDAPv3 Equivalent
ldap_search()	ldap_search_ext()
ldap_search_s()	ldap_search_ext_s()
ldap_search_st()	ldap_search_ext_s() - use timeout argument
ldap_unbind()	ldap_unbind_ext() replaces both forms which are identical besides the suffix.
ldap_unbind_s()	

Functions Alphabetically

ber_alloc()

The `ber_alloc()` function is used to allocate a new Basic Encoding Rules (BER) `BerElement` structure.

Syntax

```
#include <lber.h>
BerElement* ber_alloc( void );
```

Parameters

This function has no parameters.

Returns

- If successful, a pointer to the newly allocated `BerElement` structure.
- If unsuccessful, a `NULL` pointer.

Description

`ber_alloc()` is used to allocate a new `BerElement` structure for encoding data with the BER.

See Also

`ber_printf()`, `BerElement`

ber_alloc_t()

The `ber_alloc_t()` function constructs and returns a `BerElement` structure.

Syntax

```
#include <lber.h>
BerElement* ber_alloc_t( int options );
```

Parameters

This function has the parameter listed in Table 17-20.

Table 17-20 ber_alloc_t() Function Parameter

Parameter	Description
options	Contains bitwise OR of options to be used when generating the encoding of the BerElement. One option is defined and must always be supplied: #define LBER_USE_DER 0x01 When this option is present, lengths will always be encoded in the minimum number of octets. This option does not cause values of sets and sequences to be rearranged in tag and byte order, so these functions are not suitable for generating DER output as defined in X.509 and X.680.

Returns

- If successful, a pointer to the newly allocated BerElement structure.
- If unsuccessful, a NULL pointer.

Description

This function is the same as ber_alloc() except it has an option.

See Also

ber_printf(), BerElement

ber_bvecfree()

The ber_bvecfree() function frees an array of berval structures.

Syntax

```
#include <lber.h>
void ber_bvecfree( struct berval **bv );
```

Parameters

This function has the parameter listed in Table 17-21.

Table 17-21 ber_bvecfree() Function Parameter

Parameter	Description
<code>bv</code>	Pointer to the array that you want to free from memory.

Description

Each `berval` in the array is freed using `ber_bvfree()`, and then the array itself is freed.

See Also

`ber_bvfree()`

ber_bvdup()

The `ber_bvdup()` function returns a copy of a `berval` structure.

Syntax

```
#include <lber.h>
berval* ber_bvdup( const struct berval *bv );
```

Parameters

This function has the parameter listed in Table 17-22.

Table 17-22 ber_bvdup() Function Parameter

Parameter	Description
<code>bv</code>	Pointer to the <code>berval</code> structure to be duplicated.

Returns

- If successful, a pointer to the newly allocated `berval` structure.
- If unsuccessful, a `NULL` pointer.

Description

The `ber_bvdup()` function returns a copy of a `berval` structure. The data referenced in the structure is also duplicated. The `bv_val` field in the returned `berval` points to a different area of memory than the `bv_val` field in the argument `berval`.

ber_bvfree()

The ber_bvfree() function frees a berval structure.

Syntax

```
#include <lber.h>
void ber_bvfree( struct berval *bv );
```

Parameters

This function has the parameter listed in Table 17-23.

Table 17-23 ber_bvfree() Function Parameter

Parameter	Description
bv	Pointer to the berval structure that you want to free from memory.

Description

The ber_bvfree() function frees a berval structure from memory. Call this function to free a berval passed back from the ldap_extended_operation_s(), ldap_parse_extended_result(), ldap_sasl_bind_s(), or ldap_parse_sasl_bind_result() functions.

See Also

ldap_extended_operation_s(), ldap_parse_extended_result(), ldap_sasl_bind_s(), ldap_parse_sasl_bind_result()

ber_dup()

The ber_dup() function returns a copy of a BerElement structure.

Syntax

```
#include <lber.h>
BerElement* ber_dup( BerElement *ber );
```

Parameters

This function has the parameters listed in Table 17-24.

Table 17-24 ber_dup() Function Parameter

Parameter	Description
ber	Pointer to the BerElement to be duplicated.

Returns

- If successful, a pointer to the newly allocated BerElement structure.
- If unsuccessful, a NULL pointer.

Description

The `ber_dup()` function returns a copy of a BerElement structure.

ber_first_element()

The `ber_first_element()` function is used to return the tag and length of the first element in a set or sequence.

Syntax

```
#include <lber.h>
unsigned long ber_first_element( BerElement *ber, unsigned long *len,
    char **last );
```

Parameters

This function has the parameters listed in Table 17-25.

Table 17-25 ber_first_element() Function Parameters

Parameter	Description
ber	Pointer to a BerElement.
len	Pointer to the address of the unsigned long which returns the length of the first element.
last	Address of the pointer to return the address of the last byte of the element.

Returns

NULL pointer is returned on error.

Description

The `ber_first_element()` function is used to return the tag and length of the first element in a set or sequence. It also returns a magic cookie parameter that should be passed to subsequent calls to `ber_next_element()`.

See Also

`ber_next_element()`

ber_flatten()

The ber_flatten() function allocates a berval structure whose contents are taken from a source BerElement structure.

Syntax

```
#include <lber.h>
int ber_flatten( BerElement *ber, struct berval **bvPtr );
```

Parameters

This function has the parameters listed in Table 17-26.

Table 17-26 ber_flatten() Function Parameters

Parameter	Description
ber	Pointer to the source BerElement.
bvPtr	Pointer to the newly allocated berval structure which must be freed using ber_bvfree().

Returns

Returns 0 on success and -1 on error.

Description

This function is usually used when encoding LDAPv3 controls or extended operations values.

See Also

ber_bvfree()

ber_free()

The ber_free() function frees a BerElement structure previously allocated with ber_alloc(), ber_alloc_t(), ber_init(), or the ldap_first_attribute() and ldap_next_attribute() search functions.

Syntax

```
#include <ldap.h>
void ber_free( BerElement *ber, int freebuf );
```

Parameters

This function has the parameters listed in Table 17-27.

Table 17-27 ber_free() Function Parameters

Parameter	Description
ber	Pointer to the BerElement structure that you want to free.
freebuf	Specifies whether or not to free the buffer in the BerElement structure.

Description

This function frees a BerElement structure, which is used to keep track of the current attribute. When you are done working with the attributes, you should free this structure from memory, if it still exists.

NOTE

To retrieve attributes from a search result entry, you need to call either the ldap_first_attribute() or ldap_next_attribute() function. When freeing structures allocated by these functions, you should specify 0 for the freebuf argument. Otherwise, it should always be set to 1.

Example

Code Example 17-1 frees the BerElement structure allocated by the ldap_first_attribute() function.

Code Example 17-1 ber_free() Code Example

```
LDAP *ld;
LDAPMessage *a, *e;
BerElement *ber;
...
for ( a = ldap_first_attribute( ld, e, &ber ); a != NULL;
      a = ldap_next_attribute( ld, e, ber ) {
    ...
    /* Retrieve the value of each attribute */
    ...
}

/* Free the BerElement when done */
if ( ber != NULL ) {
    ber_free( ber, 0 );
}
...
```

ber_get_boolean()

See Also
ber_alloc(), ber_alloc_t(), ber_init(), ldap_first_attribute(),
ldap_next_attribute()

ber_get_boolean()

The ber_get_boolean() function is used to read a boolean value.

Syntax
#include <lber.h>
unsigned long ber_get_boolean(BerElement *ber,
int *boolval);

Parameters
This function has the parameters listed in Table 17-28.

Table 17-28 ber_get_boolean() Function Parameters

Parameter	Description
ber	Pointer to the BerElement structure that contains the boolean.
boolval	Specifies the boolean to read.

Returns
The value is returned on success and LBER_ERROR (-1) on failure.

ber_get_int()

The ber_get_int() function tries to interpret the next element as an integer, returning the result in num.

Syntax
#include <lber.h>
unsigned long ber_get_int(BerElement *ber, int *num);

Parameters
This function has the parameters listed in Table 17-29.

Table 17-29 ber_get_int() Function Parameters

Parameter	Description
ber	Pointer to the BerElement structure that contains the boolean.
num	Pointer to the result.

Returns

The tag of whatever it finds is returned on success and LBER_ERROR (-1) on failure.

ber_get_next()

The ber_get_next() function reads the next BER element.

Syntax

```
#include <lber.h>
unsigned long ber_get_next( Sockbuf *sb, unsigned long *len,
    BerElement *ber );
```

Parameters

This function has the parameters listed in Table 17-27.

Table 17-30 ber_get_next() Function Parameters

Parameter	Description
sb	Descriptor (socket or file descriptor) from which to read.
len	The length of the entire element.
ber	Pointer to the BerElement structure.

Description

ber_get_next() strips off and returns the leading tag, strips off and returns the length of the entire element, and sets up a pointer to **ber** for subsequent calls to decode the element.

ber_get_next_buffer()

The `ber_get_next_buffer()` function reads the next BER element from a byte buffer.

NOTE This is an older function included in the SDK for backward-compatibility. If you are writing a new LDAP client, use `ber_get_next_buffer_ext()` instead.

ber_get_next_buffer_ext()

The `ber_get_next_buffer_ext()` function reads the next BER element from a byte buffer.

NOTE `ber_get_next_buffer_ext()` is a new version of the `ber_get_next_buffer()` function. If you are writing a new LDAP client, use `ber_get_next_buffer_ext()`.

Syntax

```
#include <lber.h>
unsigned long ber_get_next_buffer_ext( void *buffer,
    size_t buffer_size, unsigned long *len, BerElement *ber,
    unsigned long *Bytes_Scanned, Sockbuf *sb );
```

Parameters

This function has the parameters listed in Table 17-31.

Table 17-31 `ber_get_next_buffer_ext()` Function Parameters

Parameter	Description
buffer	Pointer to the buffer.
buffer_size	The size of the buffer.
len	The length of the entire element.
ber	Pointer to a <code>BerElement</code> .
Bytes_Scanned	Returns the number of bytes actually searched through.
sb	Descriptor (socket or file descriptor) from which to read.

ber_get_null()

The `ber_get_null()` function is used to read a NULL element.

Syntax

```
#include <lber.h>
unsigned long ber_get_null( BerElement *ber );
```

Parameters

This function has the parameter listed in Table 17-32.

Table 17-32 `ber_get_null()` Function Parameter

Parameter	Description
ber	Pointer to a BerElement.

Returns

Returns the tag of the element it skips over.

ber_get_option()

The `ber_get_option()` function is used to retrieve information about the API and about the specific implementation being used.

Syntax

```
#include <lber.h>
int ber_get_option( BerElement *ber, int option, void *value );
```

ber_get_stringa()

The `ber_get_stringa()` function is used to allocate memory space into which an octet string is read.

Syntax

```
#include <lber.h>
unsigned long ber_get_stringa( BerElement *ber, char **buf );
```

ber_get_stringal()

The ber_get_stringal() function is used to allocate memory space into which an octet string and its length are read.

Syntax

```
#include <lber.h>
unsigned long ber_get_stringal( BerElement *ber, struct berval **bv );
```

Description

It takes a berval structure, and returns the result in this parameter.

ber_get_stringb()

The ber_get_stringb() function is used to read an octet string into a preallocated buffer.

Syntax

```
#include <lber.h>
unsigned long ber_get_stringb( BerElement *ber, char *buf,
    unsigned long *len );
```

Description

The len parameter should be initialized to the size of the buffer, and will contain the length of the octet string read upon return. The buffer should be big enough to take the octet string value plus a terminating NULL byte.

ber_init()

The ber_init() function constructs a new BerElement.

Syntax

```
#include <lber.h>
BerElement * ber_init( const struct berval *bv );
```

Parameters

This function has the parameter listed in Table 17-32.

Table 17-33 ber_init() Function Parameter

Parameter	Description
bv	Pointer to a berval.

Returns

- If successful, returns a new BerElement containing a copy of the data in the bv argument.
- If not, returns a NULL pointer.

ber_init_w_nullchar()

The ber_init_w_nullchar() function constructs a new BerElement with a NULL character.

Syntax

```
#include <lber.h>
void ber_init_w_nullchar( BerElement *ber, int options );
```

ber_next_element()

The ber_next_element() function is used to return the tag and length of the next element in a set or sequence.

Syntax

```
#include <lber.h>
unsigned long ber_next_element( BerElement *ber, unsigned long *len,
    char *last );
```

Parameters

This function has the parameters listed in Table 17-34.

Table 17-34 ber_next_element() Function Parameters

Parameter	Description
ber	Pointer to a BerElement.
len	Pointer to the address of the unsigned long which returns the length of the next element.
last	Address of the pointer to return the address of the last byte of the element.

Returns

NULL pointer is returned on error.

Description

The `ber_next_element()` function is used to return the tag and length of the first element in a set or sequence.

See Also

`ber_first_element()`

ber_peek_tag()

The `ber_peek_tag()` function returns the tag of the next element to be parsed in the supplied `BerElement` structure.

Syntax

```
#include <lber.h>
unsigned long ber_peek_tag( BerElement *ber, unsigned long *len );
```

Parameters

This function has the parameters listed in Table 17-35.

Table 17-35 `ber_peek_tag()` Function Parameters

Parameter	Description
ber	Pointer to a <code>BerElement</code> .
len	Pointer to the address of the unsigned long which returns the length of the next element.

Returns

Returns the tag of the next element to be read in the `BerElement` structure. `LBER_DEFAULT` is returned if there is no further data to be read.

ber_printf()

The `ber_printf()` function encodes a BER element.

Syntax

```
#include <lber.h>
ber_printf( BerElement *ber, const char *fmt, ... );
```

Parameters

This function has the parameters listed in Table 17-36.

Table 17-36 ber_printf() Function Parameters

Parameter	Description
ber	Pointer to a BerElement returned by ber_alloc() or ber_alloc_t().
fmt	Defines the encoding format string. The format string can contain the following characters: • -b Boolean. An integer parameter should be supplied. A boolean element is output. • -i Integer. An integer parameter should be supplied. An integer element is output. • -B Bitstring. A char * pointer to the start of the bitstring is supplied, followed by the number of bits in the bitstring. A bitstring element is output. • -n Null. No parameter is required. A NULL element is output. • -o Octet string. A char * is supplied, followed by the length of the string pointed to. An octet string element is output. • -s Octet string. A NULL-terminated string is supplied. An octet string element is output, not including the trailing NULL octet. • -t Tag. An int specifying the tag to give the next element is provided. This works across calls. • -v Several octet strings. A NULL-terminated array of char *'s is supplied. Note that a construct like '{v}' is required to get an actual sequence of octet strings. • -{ Begin sequence. No parameter is required. • -} End sequence. No parameter is required. • -[Begin set. No parameter is required. • -] End set. No parameter is required.
...	Values to be encoded.

Description

State information is kept with the ber parameter so multiple calls can be made to ber_printf() to append things to the end of the BerElement. ber_printf() interprets and formats its arguments according to the format string fmt. Each character in fmt refers to an argument to ber_printf().

Returns

-1 if there is an error during encoding.

ber_put_bitstring()

ber_put_bitstring()

The `ber_put_bitstring()` function writes a bitstring value to the given `BerElement`.

Syntax

```
#include <lber.h>
int ber_put_bitstring( BerElement *ber, char *str,
    unsigned long bitlen, unsigned long tag );
```

Description

The `ber_put_bitstring()` function writes `bitlen` bits starting at `str` as a bitstring value to the given `BerElement`.

ber_put_boolean()

The `ber_put_boolean()` function writes a boolean value to a `BerElement`.

Syntax

```
#include <lber.h>
int ber_put_boolean( BerElement *ber, int boolval, unsigned long tag );
```

Description

The boolean value is given by `boolval` to the `BerElement`.

ber_put_enum()

The `ber_put_enum()` function writes an enumerated value to a `BerElement`.

Syntax

```
#include <lber.h>
int ber_put_enum( BerElement *ber, long num, unsigned long tag );
```

ber_put_int()

The `ber_put_int()` function writes an integer to a `BerElement`.

Syntax

```
#include <lber.h>
int ber_put_int( BerElement *ber, long num, unsigned long tag );
```

ber_put_null()

The `ber_put_null()` function writes a NULL element to a `BerElement`.

Syntax

```
#include <lber.h>
int ber_put_null( BerElement *ber, unsigned long tag );
```

ber_put_ostring()

The `ber_put_ostring()` function writes bytes to a `BerElement` as an octet string.

Syntax

```
#include <lber.h>
int ber_put_ostring( BerElement *ber, char *str,
    unsigned long len, unsigned long tag );
```

ber_put_seq()

The `ber_put_seq()` function puts a sequence to a `BerElement`.

Syntax

```
#include <lber.h>
int ber_put_seq( BerElement *ber );
```

Parameters

This function has the parameter listed in Table 17-37.

Table 17-37 `ber_put_seq()` Function Parameter

Parameter	Description
<code>ber</code>	Pointer to a <code>BerElement</code> .

Description

The `ber_start_seq()` function is used to start a sequence in the `BerElement`. The end of the sequence is marked by the nearest matching call to `ber_put_seq()`.

ber_put_set()

The ber_put_set() function puts a set to a BerElement.

Syntax

```
#include <lber.h>
int ber_put_set( BerElement *ber );
```

Parameters

This function has the parameter listed in Table 17-38.

Table 17-38 ber_put_set() Function Parameter

Parameter	Description
ber	Pointer to a BerElement.

Description

The ber_start_set() function is used to start a set in the BerElement. The end of the set is marked by the nearest matching call to ber_put_set().

ber_put_string()

The ber_put_string() function writes a NULL-terminated string (minus the terminating 0) to a BerElement as an octet string.

Syntax

```
#include <lber.h>
int ber_put_string( BerElement *ber, char *str, unsigned long tag );
```

ber_read()

Syntax

```
#include <lber.h>
long ber_read( BerElement *ber, char *buf, unsigned long len );
```

ber_reset()

Syntax

```
#include <lber.h>
void ber_reset( BerElement *ber, int was_writing );
```

ber_scanf()

The `ber_scanf()` function decodes a `BerElement`.

Syntax

```
#include <lber.h>
unsigned long ber_scanf( BerElement *ber, const char *fmt, ... );
```

Parameters

This function has the parameters listed in Table 17-39.

Table 17-39 `ber_scanf()` Function Parameters

Parameter	Description
<code>ber</code>	Pointer to a <code>BerElement</code> as returned by <code>ber_get_next()</code> .

Table 17-39 ber_scanf() Function Parameters (*Continued*)

Parameter	Description
fmt	Defines the encoding format string. The format string can contain the following characters: • -a Octet string. A char ** should be supplied. Memory is allocated, filled with the contents of the octet string, NULL-terminated, and returned in the parameter. • -s Octet string. A char * buffer should be supplied, followed by a pointer to an integer initialized to the size of the buffer. Upon return, the NULL-terminated octet string is put into the buffer, and the integer is set to the actual size of the octet string. • -O Octet string. A struct ber_val ** should be supplied, which upon return points to a memory allocated struct berval containing the octet string and its length. Call ber_bvfree() to free allocated memory. • -b Boolean. A pointer to an integer should be supplied. • -i Integer. A pointer to an integer should be supplied. • -B Bitstring. A char ** should be supplied which will point to the memory allocated bits, followed by an unsigned long *, which will point to the length (in bits) of the bitstring returned. • -n Null. No parameter is required. The element is simply skipped if it is recognized. • -v Sequence of octet strings. A char *** should be supplied, which upon return points to a memory allocated NULL-terminated array of char **'s containing the octet strings. NULL is returned if the sequence is empty. • -V Sequence of octet strings with lengths. A struct berval *** should be supplied, which upon return points to a memory allocated, NULL-terminated array of struct berval **'s containing the octet strings and their lengths. NULL is returned if the sequence is empty. ber_bvecfree() can be called to free the allocated memory. • -x Skip element. The next element is skipped. • -{ Begin sequence. No parameter is required. The initial sequence tag and length are skipped. • -} End sequence. No parameter is required and no action is taken. • -[Begin set. No parameter is required. The initial set tag and length are skipped. • -] End set. No parameter is required and no action is taken.
...	Values to be encoded.

Description

`ber_scanf()` reads from `ber`, interprets the bytes according to the format string `fmt`, and stores the results in its additional arguments. The format string contains conversion specifications which are used to direct the interpretation of the BER element.

See Also

`ber_printf()`

ber_set_option()

Syntax

```
#include <lber.h>
int ber_set_option( BerElement *ber, int option, void *value );
```

ber_set_string_translators()

Syntax

```
#include <lber.h>
unsigned long ber_skip_tag( BerElement *ber, unsigned long *len );
```

ber_skip_tag()

The `ber_skip_tag()` function advances the pointer to a `BerElement` past the first tag and length to the next tag.

Syntax

```
#include <lber.h>
unsigned long ber_skip_tag( BerElement *ber, unsigned long *len );
```

Parameters

This function has the parameters listed in Table 17-40.

Table 17-40 `ber_skip_tag()` Function Parameters

Parameter	Description
<code>ber</code>	Pointer to a <code>BerElement</code> .
<code>len</code>	Pointer to the length of the value to be skipped.

`ber_sockbuf_alloc()`

Description

`ber_skip_tag()` should only be used with constructed types and situations when a BER encoding is used as the value of an octet string.

See Also

`ber_peek_tag()`

`ber_sockbuf_alloc()`

Syntax

```
#include <lber.h>
Sockbuf* ber_sockbuf_alloc( void );
```

`ber_sockbuf_free()`

Syntax

```
#include <lber.h>
void ber_sockbuf_free( Sockbuf* p );
```

`ber_sockbuf_free_data()`

Syntax

```
#include <lber.h>
void ber_sockbuf_free_data(Sockbuf *p);
```

`ber_sockbuf_get_option()`

Syntax

```
#include <lber.h>
int ber_sockbuf_get_option( Sockbuf *sb, int option, void *value );
```

ber_sockbuf_set_option()

Syntax

```
#include <lber.h>
int ber_sockbuf_set_option( Sockbuf *sb, int option, void *value );
```

ber_special_alloc()

The `ber_special_alloc()` function allocates a `BerElement` structure plus some extra memory.

Syntax

```
#include <lber.h>
void* ber_special_alloc( size_t size, BerElement **ppBer );
```

Returns

Returns a pointer to the `BerElement` and a pointer to the extra memory.

Description

`ber_special_alloc()` allocates a ber data buffer within the same block, thus saving a call to `calloc` later when we read data.

ber_special_free()

The `ber_special_free()` function frees an allocated `BerElement` structure.

Syntax

```
#include <lber.h>
void ber_special_free( void* buf, BerElement *ber );
```

See Also

`ber_special_alloc()`

ber_stack_init()

Syntax

```
#include <lber.h>
int) LDAP_CALL ber_stack_init(BerElement *ber, int options,
    char * buf, size_t size);
```

ber_start_seq()

ber_start_seq()

The `ber_start_seq()` function is used to start a sequence in a `BerElement`.

Syntax

```
#include <lber.h>
int ber_start_seq( BerElement *ber, unsigned long tag );
```

See Also

`ber_put_seq()`

ber_start_set()

The `ber_start_set()` function is used to start a set in a `BerElement`.

Syntax

```
#include <lber.h>
int ber_start_set( BerElement *ber, unsigned long tag );
```

See Also

`ber_put_set()`

ber_svecfree()

Syntax

```
#include <lber.h>
void ber_svecfree( char **vals );
```

ber_write()

Syntax

```
#include <lber.h>
long ber_write( BerElement *ber, char *buf,
               unsigned long len, int nosos );
```

ldap_abandon()

The `ldap_abandon()` function cancels an asynchronous LDAP operation that is in progress.

NOTE This is an older function included in the SDK for backward-compatibility. If you are writing a new LDAP client, use `ldap_abandon_ext()` instead.

Syntax

```
#include <ldap.h>
int ldap_abandon( LDAP *ld, int msgid );
```

Parameters

This function has the parameters listed in Table 17-41.

Table 17-41 ldap_abandon() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
msgid	Message ID of an LDAP operation.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- -1 if unsuccessful.

NOTE The appropriate LDAP error code is also set in the LDAP structure. You can retrieve the error code by calling `ldap_get_lderrno()`. Some possible codes are:

- LDAP_PARAM_ERROR if any of the arguments are invalid.
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the request.
- LDAP_SERVER_DOWN if the LDAP server did not receive the request or if the connection to the server was lost.
- LDAP_NO_MEMORY if memory cannot be allocated.

Description

Please use the newer version of this function, `ldap_abandon_ext()`.

Example

Code Example 17-2 cancels an `ldap_url_search()` operation, abandoning the results of the operation.

Code Example 17-2 Cancelling an `ldap_url_search()` Operation

```
LDAP *ld;
char *url = "ldap://ldap.itd.umich.edu/c=US?o,description? one?o=umich";
int msgid;
...
/* Initiate a search operation */
msgid = ldap_url_search( ld, url, 0 );
...
/* Abandon the search operation */
if ( ldap_abandon( ld, msgid ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_abandon" );
    return( 1 );
}
...
```

See Also

`ldap_abandon_ext()`

ldap_abandon_ext()

The `ldap_abandon_ext()` function cancels an asynchronous LDAP operation that is in progress.

NOTE `ldap_abandon_ext()` is a new version of the `ldap_abandon()` function. If you are writing a new LDAP client, use `ldap_abandon_ext()`.

Syntax

```
#include <ldap.h>
int ldap_abandon_ext( LDAP *ld, int msgid,
    LDAPControl **serverctrls, LDAPControl **clientctrls );
```

Parameters

This function has the parameters listed in Table 17-42.

Table 17-42 ldap_abandon_ext() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
msgid	Message ID of an LDAP operation to identify the operation to be cancelled. When you call an asynchronous function (such as ldap_search_ext()), the msgidp argument returns a pointer to a message ID that uniquely identifies the operation. Thus, when you call ldap_abandon_ext(), your LDAP client is able to send a request specifying the message ID of the operation to be cancelled.
serverctrls	Pointer to an array of LDAPControl structures representing LDAP server controls that apply to this operation. If you do not want to pass any server controls, specify NULL for this argument.
clientctrls	Pointer to an array of LDAPControl structures representing LDAP client controls that apply to this operation. If you do not want to pass any client controls, specify NULL for this argument.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if any of the arguments are invalid.
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the request.
- LDAP_SERVER_DOWN if the LDAP server did not receive the request or if the connection to the server was lost.
- LDAP_NO_MEMORY if memory cannot be allocated.

Description

The ldap_abandon_ext() function cancels an asynchronous LDAP operation that is in progress. For example, if you called ldap_search_ext() to initiate an LDAP search operation on the server, you can call ldap_abandon_ext() to cancel it. When you call ldap_abandon_ext(), the function checks to see if the results of the operation have already been returned. If so, ldap_abandon_ext() deletes the message ID from the queue of pending messages. If the results have not been returned, ldap_abandon_ext() sends a request to abandon the operation. Once

cancelled, the results of the operation will not be returned, even if you subsequently call `ldap_result()` to retrieve them. To identify the operation to be cancelled, specify the message ID of the operation in the `msgid` argument of the `ldap_abandon_ext()` function.

For more information, see “To Cancel an Operation in Progress” on page 82 of Chapter 4, “Using the LDAP API.”

Example

Code Example 17-3 cancels an `ldap_url_search()` operation, abandoning the results of the operation.

Code Example 17-3 `ldap_abandon_ext()` Code Example

```
LDAP *ld;
char *url = "ldap://ldap.itd.umich.edu/c=US?o,description?one?o=umich";
int msgid;
LDAPControl **srvctrls, **clntctrls;
...
/* Initiate a search operation */
msgid = ldap_url_search( ld, url, 0 );
...
/* Abandon the search operation */
if ( ldap_abandon_ext( ld, msgid, srvctrls, clntctrls )
    != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_abandon" );
    return( 1 );
}
...
```

See Also

`ldap_add_ext()`, `ldap_compare_ext()`, `ldap_delete_ext()`,
`ldap_extended_operation()`, `ldap_modify_ext()`, `ldap_rename()`,
`ldap_sasl_bind()`, `ldap_search_ext()`, `ldap_simple_bind()`, `ldap_url_search()`

ldap_add()

The `ldap_add()` function adds a new entry to the directory asynchronously.

NOTE

This is an older function that is included in the SDK for backward-compatibility. If you are writing a new LDAP client, use `ldap_add_ext()` instead.

Syntax

```
#include <ldap.h>
int ldap_add( LDAP *ld, const char *dn, LDAPMod **attrs );
```

Parameters

This function has the parameters listed in Table 17-43.

Table 17-43 ldap_add() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
dn	DN of the entry to add. With the exception of the leftmost component, all components of the DN (for example, o=organization or c=country) must already exist.
attrs	Pointer to a NULL-terminated array of pointers to LDAPMod structures representing the attributes of the new entry.

Returns

The message ID of the ldap_add() operation.

NOTE	To check the result of this operation, call ldap_result() and ldap_result2error(). See ldap_add_ext_s() for a list of possible result codes for the LDAP add operation.

Description

Please use the newer version of this function, ldap_add_ext().

Example

Code Example 17-4 adds a new entry to the directory.

Code Example 17-4 ldap_add() Code Example

```
#include <ldap.h>
...
LDAP *ld;
LDAPMod *list_of_attrs[4];
LDAPMod attribute1, attribute2, attribute3;
LDAPMessage *result;
int msgid, rc;
struct timeval tv;
```

Code Example 17-4 ldap_add() Code Example (*Continued*)

```

/* Distinguished name of the new entry. Note that "dc=example,dc=com" and
"ou=People,dc=example,dc=com" must already exist in the directory. */
char *dn = "uid=bjensen,ou=People,dc=example,dc=com";

/* To add a "person" entry, you must specify values for the sn, cn, and
objectClass attributes. (These are required attributes.) */
char *sn_values[] = { "Jensen", NULL };

/* To specify multiple values for an attribute, add the different values to
the array. */
char *cn_values[] = { "Barbara Jensen", "Babs Jensen", NULL };

/* The object class for a "person" entry is "inetOrgPerson", which is a
subclass of "top", "person", and "organizationalPerson". You should add all
of these classes as values of the objectClass attribute. */
char *objectClass_values[] = { "top", "person", "organizationalPerson",
"inetOrgPerson", NULL };

...
/* Specify the value and type of each attribute in separate LDAPMod
structures */
attribute1.mod_type = "sn";
attribute1.mod_values = sn_values;
attribute2.mod_type = "cn";
attribute2.mod_values = cn_values;
attribute3.mod_type = "objectClass";
attribute3.mod_values = objectClass_values;

/* Add the pointers to these LDAPMod structures to an array */
list_of_attrs[0] = &attribute1;
list_of_attrs[1] = &attribute2;
list_of_attrs[2] = &attribute3;
list_of_attrs[3] = NULL;

...
/* Set up the timeout period for adding the new entry */
tv.tv_sec = tv.tv_usec = 0;

/* Add the user "Barbara Jensen" */
if ( ( msgid = ldap_add( ld, dn, list_of_attrs ) ) == -1 ) {
    ldap_perror( ld, "ldap_add" );
    return( 1 );
}

/* Check to see if the operation has completed */
while ( ( rc = ldap_result( ld, msgid, 0, &tv, &result ) ) == 0 ) {
    ...
    /* do other work while waiting for the operation to complete */
    ...
}

/* Check the result to see if any errors occurred */
if ( ( rc = ldap_result2error( ld, result, 1 ) ) != LDAP_SUCCESS ) {
    printf( "Error while adding entry: %s\n", ldap_err2string( rc ) );
}

```

Code Example 17-4 ldap_add() Code Example (Continued)

```

}
...

```

See Also

ldap_add_ext()

ldap_add_ext()

The ldap_add_ext() function adds a new entry to the directory asynchronously.

NOTE ldap_add_ext() is a new version of the ldap_add() function. If you are writing a new LDAP client, use ldap_add_ext().

Syntax

```

#include <ldap.h>
int ldap_add_ext( LDAP *ld, const char *dn, LDAPMod **attrs,
                 LDAPControl **serverctrls, LDAPControl **clientctrls,
                 int *msgidp );

```

Parameters

This function has the parameters listed in Table 17-44.

Table 17-44 ldap_add_ext() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
dn	DN of the entry to add. With the exception of the leftmost component, all components of the DN (for example, o=organization or c=country) must already exist.
attrs	Pointer to a NULL-terminated array of pointers to LDAPMod structures representing the attributes of the new entry.
serverctrls	Pointer to an array of LDAPControl structures representing LDAP server controls that apply to this operation. If you do not want to pass any server controls, specify NULL for this argument.

Table 17-44 ldap_add_ext() Function Parameters (*Continued*)

Parameter	Description
clientctrls	Pointer to an array of LDAPControl structures representing LDAP client controls that apply to this operation. If you do not want to pass any client controls, specify NULL for this argument.
msgidp	Pointer to an integer that will be set to the message ID of the LDAP operation.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if any of the arguments are invalid.
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the request.
- LDAP_SERVER_DOWN if the LDAP server did not receive the request or if the connection to the server was lost.
- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_NOT_SUPPORTED if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3. Make sure that you set the version of your LDAP client to version 3 before calling this function. For details, see “Specifying the LDAP Version of Your Client” on page 65 of Chapter 3, “Writing an LDAP Client.”

Description

The ldap_add_ext() adds a new entry to the directory. To add a new entry to the directory, you need to specify:

- A unique DN identifying the new entry.
Use the dn argument to specify the DN of the new entry. The parents of the entry should already exist. For example, if you are adding the entry uid=bjensen,ou=People,dc=example,dc=com, the entries ou=People and dc=example,dc=com must already exist in the directory.
- A set of attributes for the new entry.
Create an LDAPMod structure for each attribute. Set the mod_op field to 0 if the attribute values are string values. To specify values that consist of binary data (such as a sound file or a JPEG file), set the mod_op field to LDAP_MOD_BVALUES. Create an array of these LDAPMod structures and pass the array as the attrs argument.

`ldap_add_ext()` is an asynchronous function; it does not directly return results. In order to get the results of the asynchronous operation, you need to call the `ldap_result()` and `ldap_parse_result()` functions. If you want the results to be returned directly by the function, call the synchronous function `ldap_add_ext_s()`. For a list of possible result codes for an LDAP add operation, see the `ldap_add_ext_s()` function.

For additional information on adding new entries to the directory, see “Adding an Entry” on page 149 of Chapter 7, “Adding, Modifying and Deleting Entries.”

Example

See Code Example 4-7 on page 82 of Chapter 4, “Using the LDAP API.”

See Also

`ldap_add_ext_s()`, `ldap_result()`, `ldap_parse_result()`, `LDAPMod`

ldap_add_ext_s()

The `ldap_add_ext_s()` function adds a new entry to the directory synchronously.

NOTE

`ldap_add_ext_s()` is a new version of the `ldap_add_s()` function. If you are writing a new LDAP client, use `ldap_add_ext_s()`.

Syntax

```
#include <ldap.h>
int ldap_add_ext_s( LDAP *ld, const char *dn, LDAPMod **attrs,
    LDAPControl **serverctrls, LDAPControl **clientctrls );
```

Parameters

This function has the parameters listed in Table 17-45.

Table 17-45 `ldap_add_ext_s()` Function Parameters

Parameter	Description
<code>ld</code>	Pointer to an LDAP structure containing information about the connection to the LDAP server.
<code>dn</code>	DN of the entry to add. With the exception of the leftmost component, all components of the DN (for example, <code>o=organization</code> or <code>c=country</code>) must already exist.
<code>attrs</code>	Pointer to a NULL-terminated array of pointers to <code>LDAPMod</code> structures representing the attributes of the new entry.

Table 17-45 ldap_add_ext_s() Function Parameters (*Continued*)

Parameter	Description
serverctrls	Pointer to an array of LDAPControl structures representing LDAP server controls that apply to this operation. If you do not want to pass any server controls, specify NULL for this argument.
clientctrls	Pointer to an array of LDAPControl structures representing LDAP client controls that apply to this operation. If you do not want to pass any client controls, specify NULL for this argument.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if any of the arguments are invalid.
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the request.
- LDAP_SERVER_DOWN if the LDAP server did not receive the request or if the connection to the server was lost.
- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_LOCAL_ERROR if an error occurred when receiving the results from the server.
- LDAP_DECODING_ERROR if an error occurred when decoding the BER-encoded results from the server.
- LDAP_NOT_SUPPORTED if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3. Make sure that you set the version of your LDAP client to version 3 before calling this function. For details, see “Specifying the LDAP Version of Your Client” on page 65 of Chapter 3, “Writing an LDAP Client.”

NOTE	Sun Java System Directory Server and other LDAP server products may send result codes in addition to those described. For example, the server may have loaded a plug-in that returns custom result codes. Check your LDAP server documentation for other result codes.

Description

The ldap_add_ext_s() function adds a new entry to the directory. To add a new entry to the directory, you need to specify:

- A unique DN identifying the new entry.

Use the `dn` argument to specify the DN of the new entry. The parents of the entry should already exist. For example, if you are adding the entry `uid=bjensen,ou=People,dc=example,dc=com`, the entries `ou=People` and `dc=example,dc=com` must already exist in the directory.

- A set of attributes for the new entry.

Create an `LDAPMod` structure for each attribute. Set the `mod_op` field to 0 if the attribute values are string values. To specify values that consist of binary data (such as a sound file or a JPEG file), set the `mod_op` field to `LDAP_MOD_BVALUES`. Create an array of these `LDAPMod` structures and pass the array as the `attrs` argument.

`ldap_add_ext_s()` is a synchronous function; it directly returns the results of the operation. If you want to perform other operations while waiting for the results of this operation, call the asynchronous function `ldap_add_ext()`.

For additional information on adding new entries to the directory, see “Adding an Entry” on page 149 of Chapter 7, “Adding, Modifying and Deleting Entries.”

Example

See Code Example 7-8 on page 154 of Chapter 7, “Adding, Modifying and Deleting Entries.”

See Also

`ldap_add_ext()`, `LDAPMod`

ldap_add_s()

The `ldap_add_s()` function adds a new entry to the directory synchronously.

NOTE

This is an older function that is included in the SDK for backward-compatibility. If you are writing a new LDAP client, use `ldap_add_ext_s()` instead.

Syntax

```
#include <ldap.h>
int ldap_add_s( LDAP *ld, const char *dn, LDAPMod **attrs );
```

Parameters

This function has the parameters listed in Table 17-46.

Table 17-46 ldap_add_s() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
dn	DN of the entry to add. With the exception of the leftmost component, all components of the DN (for example, o=organization or c=country) must already exist.
attrs	Pointer to a NULL-terminated array of pointers to LDAPMod structures representing the attributes of the new entry.

Returns

See ldap_add_ext_s() for a list of possible return codes.

Description

Please use the newer version of this function, ldap_add_ext_s().

Example

Code Example 17-5 adds a new entry to the directory.

Code Example 17-5 ldap_add_s Code Example

```
#include <ldap.h>
...
LDAP *ld;
LDAPMod *list_of_attrs[4];
LDAPMod attribute1, attribute2, attribute3;

/* Distinguished name of the new entry. Note that "dc=example,dc=com" and
"ou=People,dc=example,dc=com" must already exist in the directory. */
char *dn = "uid=bjensen,ou=People,dc=example, dc=com";

/* To add a "person" entry, you must specify values for the sn, cn, and
objectClass attributes. (These are required attributes.) */
char *sn_values[] = { "Jensen", NULL };

/* To specify multiple values for an attribute, add the different values to
the array. */
char *cn_values[] = { "Barbara Jensen", "Babs Jensen", NULL };

/* The object class for a "person" entry is "inetOrgPerson", which is a
subclass of "top", "person", and "organizationalPerson". You should add all
of these classes as values of the objectClass attribute. */
char *objectClass_values[] = { "top", "person", "organizationalPerson",
"inetOrgPerson", NULL };
```


Code Example 17-5 ldap_add_s Code Example (Continued)

```

...
/* Specify the value and type of each attribute in separate LDAPMod
structures */
attribute1.mod_type = "sn";
attribute1.mod_values = sn_values;
attribute2.mod_type = "cn";
attribute2.mod_values = cn_values;
attribute3.mod_type = "objectClass";
attribute3.mod_values = objectClass_values;

/* Add the pointers to these LDAPMod structures to an array */
list_of_attrs[0] = &attribute1;
list_of_attrs[1] = &attribute2;
list_of_attrs[2] = &attribute3;
list_of_attrs[3] = NULL;
...
/* Add the user "Barbara Jensen" */
if ( ldap_add_s( ld, dn, list_of_attrs ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_add_s" );
    return( 1 );
}
...

```

See Also

ldap_add_ext_s()

ldap_ber_free()

The ldap_ber_free() function frees a BerElement structure from memory.

NOTE

This is an older function that is included in the SDK for backward-compatibility. If you are writing a new LDAP client, use ber_free() instead. Except in name, the two functions are identical.

Syntax

```

#include <ldap.h>
void ldap_ber_free( BerElement *ber, int freebuf );

```

Parameters

This function has the parameters listed in Table 17-47.

Table 17-47 ldap_ber_free() Function Parameters

Parameter	Description
ber	Pointer to the BerElement structure that you want to free.
freebuf	Specifies whether or not to free the buffer in the BerElement structure.

Description

Please use the newer version of this function, ber_free().

See Also

ber_free()

ldap_bind()

CAUTION This function is deprecated and should not be used. It is included in ldap-deprecated.h for backward-compatibility. Please use ldap_simple_bind().

ldap_bind() can be used when the authentication method being used needs to be selected at runtime.

Syntax

```
#include <ldap-deprecated.h>
ldap_bind( LDAP *ld, const char *who, const char *passwd, int method );
```

Parameters

This function has the parameters listed in Table 17-48.

Table 17-48 ldap_bind() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
who	DN of the user who wants to authenticate. For anonymous authentication, set this or the passwd argument to NULL.
passwd	Password of the user who wants to authenticate. For anonymous authentication, set this or the who argument to NULL.

Table 17-48 ldap_bind() Function Parameters (*Continued*)

Parameter	Description
method	Defines the authentication method to be used. It should be set to LDAP_AUTH_SIMPLE to select simple authentication.

Returns

ldap_bind() returns the message ID of the request it initiates.

Description

This is an asynchronous function that authenticates a specified entry to the directory. After a connection is made to an LDAP server using ldap_open(), a bind operation must be performed before other operations can be attempted over the connection.

See Also

ldap_open(), ldap_init(), ldap_bind_s()

ldap_bind_s()

CAUTION This function is deprecated and should not be used. It is included in ldap-deprecated.h for backward-compatibility. Please use ldap_simple_bind_s().

ldap_bind_s() can be used when the authentication method needs to be selected at runtime.

Syntax

```
#include <ldap-deprecated.h>
ldap_bind_s( LDAP *ld, const char *who, const char *cred, int method );
```

Parameters

This function has the parameters listed in Table 17-49.

Table 17-49 ldap_bind_s() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.

Table 17-49 ldap_bind_s() Function Parameters (*Continued*)

Parameter	Description
who	DN of the user who wants to authenticate. For anonymous authentication, set this or the passwd argument to NULL.
cred	Password of the user who wants to authenticate. For anonymous authentication, set this or the who argument to NULL.
method	Defines the authentication method to be used. It should be set to LDAP_AUTH_SIMPLE to select simple authentication.

Returns

ldap_bind_s() returns an LDAP error indication.

Description

This is a synchronous function that authenticates a specified entry to the directory. After a connection is made to an LDAP server using ldap_open(), a bind operation must be performed before other operations can be attempted over the connection.

See Also

ldap_open(), ldap_init(), ldap_bind()

ldap_build_filter()

CAUTION

This function is deprecated and should not be used. It is included in ldap-deprecated.h for backward-compatibility. Please use ldap_create_filter().

See Also

ldap_create_filter()

ldap_cache_flush()

CAUTION

This function is deprecated and should not be used. It is included in ldap-deprecated.h for backward-compatibility. Please use the ldap_memcache_* functions.

For more specific information, see the header file ldap-deprecated.h.

See Also

ldap_memcache_init(), ldap_memcache_set(), ldap_memcache_get(),
 ldap_memcache_update(), ldap_memcache_flush(), ldap_memcache_destroy()

ldap_charray_add()

CAUTION This function is deprecated and should not be used. It is included in ldap-to-be-deprecated.h for backward-compatibility.

For more specific information, see the header file ldap-to-be-deprecated.h.

ldap_charray_dup()

CAUTION This function is deprecated and should not be used. It is included in ldap-to-be-deprecated.h for backward-compatibility.

For more specific information, see the header file ldap-to-be-deprecated.h.

ldap_charray_free()

CAUTION This function is deprecated and should not be used. It is included in ldap-to-be-deprecated.h for backward-compatibility.

For more specific information, see the header file ldap-to-be-deprecated.h.

ldap_charray_inlist()

CAUTION This function is deprecated and should not be used. It is included in ldap-to-be-deprecated.h for backward-compatibility.

For more specific information, see the header file ldap-to-be-deprecated.h.

ldap_charray_merge()

CAUTION

This function is deprecated and should not be used. It is included in `ldap-to-be-deprecated.h` for backward-compatibility.

For more specific information, see the header file `ldap-to-be-deprecated.h`.

ldap_charray_position()

CAUTION

This function is deprecated and should not be used. It is included in `ldap-to-be-deprecated.h` for backward-compatibility.

For more specific information, see the header file `ldap-to-be-deprecated.h`.

ldap_compare()

The `ldap_compare()` function asynchronously determines if an attribute of an entry contains a specified value.

NOTE

This is an older function that is included in the SDK for backward-compatibility. If you are writing a new LDAP client, use `ldap_compare_ext()` instead.

Syntax

```
#include <ldap.h>
int ldap_compare( LDAP *ld, const char *dn, const char *attr,
    const char *value );
```

Parameters

This function has the parameters listed in Table 17-50.

Table 17-50 ldap_compare() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
dn	DN of the entry used in the comparison.

Table 17-50 ldap_compare() Function Parameters (*Continued*)

Parameter	Description
attr	Attribute type that you want to check the value against.
value	Value that you want to compare against the attribute values.

Returns

Returns the message ID of the ldap_compare() operation.

NOTE

To check the result of this operation, call ldap_result() and ldap_result2error(). See ldap_compare_ext_s() function for a list of possible result codes for the LDAP compare operation.

Description

Please use the newer version of this function, ldap_compare_ext().

Example

Code Example 17-6 checks to see if Barbara Jensen has the e-mail address *bjensen@example.com*.

Code Example 17-6 ldap_compare() Code Sample

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
char *dn = "uid=bjensen,ou=People,dc=example, dc=com";
int msgid;
...
msg_id = ldap_compare( ld, dn, "mail", "bjensen@example.com" );
...
```

See Also

ldap_compare_ext()

ldap_compare_ext()

The ldap_compare_ext() function asynchronously determines if an attribute of an entry contains a specified value.

NOTE ldap_compare_ext() is a new version of the ldap_compare() function. If you are writing a new LDAP client, use ldap_compare_ext().

Syntax

```
#include <ldap.h>
int ldap_compare_ext( LDAP *ld, const char *dn, const char *attr,
    struct berval *bvalue, LDAPControl **serverctrls,
    LDAPControl **clientctrls, int *msgidp );
```

Parameters

This function has the parameters listed in Table 17-51.

Table 17-51 ldap_compare_ext() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
dn	DN of the entry used in the comparison.
attr	Attribute type that you want to check the value against.
value	Value that you want to compare against the attribute values.
serverctrls	Pointer to an array of LDAPControl structures representing LDAP server controls that apply to this operation. If you do not want to pass any server controls, specify NULL for this argument.
clientctrls	Pointer to an array of LDAPControl structures representing LDAP client controls that apply to this operation. If you do not want to pass any client controls, specify NULL for this argument.
msgidp	Pointer to an integer that will be set to the message ID of the LDAP operation.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if any of the arguments are invalid.
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the request.
- LDAP_SERVER_DOWN if the LDAP server did not receive the request or if the connection to the server was lost.

- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_NOT_SUPPORTED if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3. Make sure that you set the version of your LDAP client to version 3 before calling this function. For details, see “Specifying the LDAP Version of Your Client” on page 65 of Chapter 3, “Writing an LDAP Client.”

NOTE To check the result of this operation, call `ldap_result()` and `ldap_parse_result()`.

Returns

For a list of the possible result codes for an LDAP compare operation, see `ldap_compare_ext_s()`.

Description

The `ldap_compare_ext()` function compares the value of an attribute in an entry against a specified value. Because `ldap_compare_ext()` is an asynchronous function, it does not directly return results. If you want the results to be returned directly by the function, call the synchronous function `ldap_compare_ext_s()`.

For additional information on comparing attribute values in an entry, see Chapter 8, “Comparing Entries.”

Example

See Code Example 8-4 on page 195 of Chapter 8, “Comparing Entries.”

See Also

`ldap_compare_ext_s()`, `ldap_result()`, `ldap_parse_result()`

ldap_compare_ext_s()

The `ldap_compare_ext_s()` function synchronously determines if an attribute of an entry contains a specified value.

Syntax

```
#include <ldap.h>
int ldap_compare_ext_s( LDAP *ld, const char *dn,
    const char *attr, struct berval *bvalue,
    LDAPControl **serverctrls, LDAPControl **clientctrls );
```

Parameters

This function has the parameters listed in Table 17-52.

Table 17-52 ldap_compare_ext_s() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
dn	DN of the entry used in the comparison.
attr	Attribute type that you want to check the value against.
value	Value that you want to compare against the attribute values.
serverctrls	Pointer to an array of LDAPControl structures representing LDAP server controls that apply to this operation. If you do not want to pass any server controls, specify NULL for this argument.
clientctrls	Pointer to an array of LDAPControl structures representing LDAP client controls that apply to this operation. If you do not want to pass any client controls, specify NULL for this argument.

Returns

One of the following values:

- LDAP_COMPARE_TRUE if the entry contains the attribute value.
- LDAP_COMPARE_FALSE if the entry does not contain the attribute value.
- LDAP_PARAM_ERROR if any of the arguments are invalid.
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the request.
- LDAP_SERVER_DOWN if the LDAP server did not receive the request or if the connection to the server was lost.
- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_LOCAL_ERROR if an error occurred when receiving the results from the server.
- LDAP_DECODING_ERROR if an error occurred when decoding the BER-encoded results from the server.

- LDAP_NOT_SUPPORTED if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3. Make sure that you set the version of your LDAP client to version 3 before calling this function. For details, see “Specifying the LDAP Version of Your Client” on page 65 of Chapter 3, “Writing an LDAP Client.”

NOTE

Sun Java System Directory Server and other LDAP server products may send result codes in addition to those described. For example, the server may have loaded a plug-in that returns custom result codes. Check your LDAP server documentation for other result codes.

Description

The `ldap_compare_ext_s()` function compares the value of an attribute in an entry against a specified value. `ldap_compare_ext_s()` is a synchronous function, which directly returns the results of the operation. If you want to perform other operations while waiting for the results of this operation, call the asynchronous function `ldap_compare_ext()`.

For additional information on comparing attribute values in an entry, see Chapter 8, “Comparing Entries.”

Example

See Code Example 8-2 on page 191 of Chapter 8, “Comparing Entries.”

See Also

`ldap_compare_ext()`

ldap_compare_s()

The `ldap_compare_s()` function synchronously determines if an attribute of an entry contains a specified value.

NOTE

This is an older function that is included in the SDK for backward-compatibility. If you are writing a new LDAP client, use `ldap_compare_ext_s()` instead.

Syntax

```
#include <ldap.h>
int ldap_compare_s( LDAP *ld, const char *dn,
    const char *attr, const char *value );
```

Parameters

This function has the parameters listed in Table 17-53.

Table 17-53 ldap_compare_s() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
dn	DN of the entry used in the comparison.
attr	Attribute type that you want to check the value against.
value	Value that you want to compare against the attribute values.

Returns

For a list of the possible result codes for an LDAP compare operation, see ldap_compare_ext_s().

Description

Please use the newer version of this function, ldap_compare_ext_s().

Example

Code Example 17-7 checks to see if Barbara Jensen has the e-mail address bjensen@example.com.

Code Example 17-7 ldap_compare_s() Code Sample

```
#include <stdio.h>
#include <ldap.h>
LDAP *ld;
char *dn = "uid=bjensen,ou=People,dc=example, dc=com";
int has_value;
...
has_value = ldap_compare_s( ld, dn, "mail", "bjensen@example.com" );
switch ( has_value ) {
    case LDAP_COMPARE_TRUE:
        printf( "The mail attribute contains bjensen@example.com.\n" );
        break;
    case LDAP_COMPARE_FALSE:
        printf( "The mail attribute does not contain bjensen@example.com.\n" );
        break;
    default:
        ldap_perror( ld, "ldap_compare_s" );
        return( 1 );
}
...
```

See Also

ldap_compare_ext_s()

ldap_control_free()

The `ldap_control_free()` function frees an `LDAPControl` structure from memory.

Syntax

```
#include <ldap.h>
void ldap_control_free( LDAPControl *ctrl );
```

Parameters

This function has the parameter listed in Table 17-54.

Table 17-54 ldap_control_free() Function Parameter

Parameter	Description
ctrl	Pointer to an <code>LDAPControl</code> structure that you want to free from memory.

Description

The `ldap_control_free()` function frees an `LDAPControl` structure from memory. You should call this function to free controls that you create; for example, if you call the `ldap_create_sort_control()` function.

See Also

ldap_controls_free()

ldap_controls_free()

The `ldap_controls_free()` function frees an array of `LDAPControl` structures from memory.

Syntax

```
#include <ldap.h>
void ldap_controls_free( LDAPControl **ctrls );
```

Parameters

This function has the parameter listed in Table 17-55.

Table 17-55 ldap_controls_free() Function Parameter

Parameter	Description
ctrls	Pointer to an array of LDAPControl structures that you want to free from memory.

Description

The ldap_controls_free() function frees an array of LDAPControl structures from memory. You should call this function to free arrays of controls that you create or any arrays returned by ldap_parse_result().

See Also

ldap_control_free(), ldap_parse_result()

ldap_count_entries()

The ldap_count_entries() function returns the number of LDAPMessage structures representing directory entries in a chain of search results.

Syntax

```
#include <ldap.h>
int ldap_count_entries( LDAP *ld, LDAPMessage *result );
```

Parameters

This function has the parameters listed in Table 17-56.

Table 17-56 ldap_count_entries() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
result	Chain of search results, represented by pointer to an LDAPMessage structure.

Returns

One of the following values:

- If successful, the number of LDAPMessage structures of the type LDAP_RES_SEARCH_ENTRY in a chain of search results. (If there are no structures of this type, returns 0.)

- -1 if ld is not a valid connection handle.

Description

The ldap_count_entries() function returns the number of LDAPMessage structures representing directory entries in a chain of search results. These messages have the type LDAP_RES_SEARCH_ENTRY.

NOTE

If you pass in a pointer to an LDAPMessage structure in the middle of the chain of results, the function counts only the entries between that structure and the last structure in the chain. In this type of situation, the function does not return the count of all entries in the chain.

For more information on using this function, see “Getting Search Results” on page 114 of Chapter 5, “Searching a LDAP Directory.”

Example

See the examples under ldap_search_ext() and ldap_search_ext_s().

See Also

ldap_result(), ldap_search_ext(), ldap_search_ext_s(), ldap_first_entry(), ldap_next_entry(), ldap_first_entry(), ldap_next_message()

ldap_count_messages()

The ldap_count_messages() function returns the number of LDAPMessage structures in a chain of search results.

Syntax

```
#include <ldap.h>
int ldap_count_messages( LDAP *ld, LDAPMessage *res );
```

Parameters

This function has the parameters listed in Table 17-57.

Table 17-57 ldap_count_messages() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
result	Chain of search results, represented by pointer to an LDAPMessage structure.

Returns

One of the following values:

- The number of LDAPMessage structures in a chain of search results, if successful. (If there are no structures, returns 0.)
- -1 if ld is not a valid connection handle.

Description

The ldap_count_messages() function returns the number of LDAPMessage structures in a chain of search results. The count is the number of search entries plus the number of search references.

NOTE

If you pass in a pointer to an LDAPMessage structure in the middle of the chain of results, the function counts only between that structure and the last structure in the chain. In this type of situation, the function does not return the count of all structures in the chain.

For more information on using this function, see “Getting Search Results” on page 114 of Chapter 5, “Searching a LDAP Directory.”

Example

See the examples under ldap_search_ext() and ldap_search_ext_s().

See Also

ldap_result(), ldap_search_ext(), ldap_search_ext_s(),
ldap_first_message(), ldap_next_message(), ldap_first_entry(),
ldap_next_entry(), ldap_first_reference(), ldap_next_reference()

ldap_count_references()

The ldap_count_references() function returns the number of LDAPMessage structures representing search references in a chain of search results.

Syntax

```
#include <ldap.h>
int ldap_count_references( LDAP *ld, LDAPMessage *res );
```

Parameters

This function has the parameters listed in Table 17-58.

Table 17-58 ldap_count_references() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
result	Chain of search results, represented by pointer to an LDAPMessage structure.

Returns

One of the following values:

- The number of LDAPMessage structures of the type LDAP_RES_SEARCH_REFERENCE in a chain of search results, if successful. (If there are no structures of this type, returns 0.)
- -1 if ld is not a valid connection handle.

Description

The ldap_count_references() function returns the number of LDAPMessage structures representing search references in a chain of search results. These messages have the type LDAP_RES_SEARCH_REFERENCE.

NOTE	If you pass in a pointer to an LDAPMessage structure in the middle of the chain of results, the function counts only the references between that structure and the last structure in the chain. In this type of situation, the function does not return the count of all references in the chain.

For more information on using this function, see “Getting Search Results” on page 114 of Chapter 5, “Searching a LDAP Directory.”

Example

See the examples under ldap_search_ext() and ldap_search_ext_s().

See Also

ldap_result(), ldap_search_ext(), ldap_search_ext_s(),
ldap_first_reference(), ldap_next_reference()

ldap_count_values()

The ldap_count_values() function returns the number of values in an array of strings.

NOTE Use the `ldap_count_values_len()` function if the array contains `berval` structures.

Syntax

```
#include <ldap.h>
int ldap_count_values( char **values );
```

Parameters

This function has the parameter listed in Table 17-59.

Table 17-59 `ldap_count_values()` Function Parameter

Parameter	Description
values	Array of values.

Returns

One of the following values:

- The number of values in the array, if successful.
- -1 if unsuccessful.

See Chapter 18, “Result Codes” for a complete listing.

Description

For additional information, see “Getting the Values of an Attribute” on page 128 of Chapter 5, “Searching a LDAP Directory.”

Example

Code Example 17-8 counts the number of values assigned to an attribute.

Code Example 17-8 `ldap_count_values()` Code Example

```
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *e;
char *a="cn";
char **vals;
int count;
...

/* Get the values of the cn attribute */
vals = ldap_get_values( ld, e, a );
```

Code Example 17-8 ldap_count_values() Code Example (*Continued*)

```
/* Count the values of the attribute */
count = ldap_count_values( vals );
...
```

See Also
ldap_count_values_len(), ldap_get_values()

ldap_count_values_len()

The ldap_count_values_len() function returns the number of values in an array of berval structures.

NOTE Use the ldap_count_values() function if the array contains strings.

Syntax
#include <ldap.h>
int ldap_count_values_len(struct berval **vals);

Parameters
This function has the parameter listed in Table 17-60.

Table 17-60 ldap_count_values_len() Function Parameter

Parameter	Description
values	Array of berval structures.

Returns
One of the following values:

- The number of values in the array, if successful.
- -1 if unsuccessful.

See Chapter 18, “Result Codes” for a complete listing.

Description

For additional information, see “Getting the Values of an Attribute” on page 128 of Chapter 5, “Searching a LDAP Directory.”

Example

Code Example 17-9 counts the number of values assigned to an attribute.

Code Example 17-9 ldap_count_values_len() Code Example

```
#include <ldap.h>
LDAP *ld;
LDAPMessage *e;
char *a="jpegPhoto";
struct berval **bvals;
int count;
...
/* Get the values of the jpegPhoto attribute */
bvals = ldap_get_values_len( ld, e, a );

/* Count the values of the attribute */
count = ldap_count_values_len( vals );
...
```

See Also

ldap_count_values(), ldap_get_values_len()

ldap_create_filter()

The ldap_create_filter() routine constructs an LDAP search filter.

Syntax

```
#include <ldap.h>
int ldap_create_filter( char *buf, unsigned long buflen,
    char *pattern, char *prefix, char *suffix, char *attr,
    char *value, char **valwords );
```

Parameters

This function has the parameters listed in Table 17-61.

Table 17-61 ldap_create_filter() Function Parameters

Parameter	Description
buf	Buffer to contain the constructed filter.
buflen	Size of the buffer.
pattern	Pattern for the filter.
prefix	Prefix to prepend to the filter. (NULL if not used.)
suffix	Suffix to append to the filter. (NULL if not used.)
attr	Replaces %a in the pattern.
value	Replaces %v in the pattern.
valwords	Replaces %vM through %vN in the pattern.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_SIZELIMIT_EXCEEDED if the created filter exceeds the size of the buffer.
- LDAP_PARAM_ERROR if an invalid parameter was passed to the function.

Description

For more information about filters, see “Creating Filters Programmatically” on page 146 of Chapter 6, “Using Filter Configuration Files.”

Example

Code Example 17-10 builds the filter (mail=bjensen@example.com).

Code Example 17-10 Creating a Filter with ldap_create_filter()

```
char buf[LDAP_FILT_MAXSIZ];
char *pattern = "(%a=%v)";
char *attr = "mail";
char *value = "bjensen@example.com";
...
ldap_create_filter( buf, LDAP_FILT_MAXSIZ, pattern, NULL,
    NULL, attr, value, NULL );
...
```

See Also

ldap_init_getfilter(), ldap_init_getfilter_buf(), ldap_getfirstfilter(),
ldap_getnextfilter(), ldap_set_filter_additions()

ldap_create_persistentsearch_control()

The ldap_create_persistentsearch_control() function creates a control that allows your client to perform a search of an LDAP v3 server that continues without termination until your client abandons it.

NOTE

Persistent search is an optional feature; it may not be supported on all LDAP servers. In order for the control to work, the server that you are connecting to must support the server control for persistent searches (OID 2.16.840.1.113730.3.4.3, or LDAP_CONTROL_PERSISTENTSEARCH, as defined in the ldap.h header file). See “Determining Supported Controls” on page 234 of Chapter 13, “Working with LDAP Controls”for information on determining the controls supported by a server.

Syntax

```
#include <ldap.h>
int ldap_create_persistentsearch_control( LDAP *ld,
    int changetypes, int changesonly, int return_echg_ctls,
    char ctl_iscritical, LDAPControl **ctrlp );
```

Parameters

This function has the parameters listed in Table 17-62.

Table 17-62 ldap_create_persistentsearch_control() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.

Table 17-62 ldap_create_persistentsearch_control() Function Parameters

Parameter	Description
changetypes	Specifies the types of changes that you want to keep track of. This field can have one or more of the following values: - LDAP_CHANGETYPE_ADD specifies that you want to keep track of entries added to the directory. - LDAP_CHANGETYPE_DELETE specifies that you want to keep track of entries deleted from the directory. - LDAP_CHANGETYPE_MODIFY specifies that you want to keep track of entries that are modified. - LDAP_CHANGETYPE_MOVDN specifies that you want to keep track of entries that are renamed. - LDAP_CHANGETYPE_ANY specifies that you want to keep track of all of the above changes to the directory. You can OR the values together to specify multiple types.
changesonly	Specifies whether or not you want skip the initial search and only get the latest changes as they occur: - If non-zero, the initial search is skipped and only entries that have changed after the initial search are returned. - If 0, the results of the initial search are returned first.
return_echg_ctrls	Specifies whether or not entry controls are included with each entry returned to your client: - If non-zero, a control is included with each entry. - If 0, controls are not included with the entries returned from the server.
ctl_iscritical	Specifies whether the control is critical to the operation. - If non-zero, the control is critical to the operation. If the server does not support it, the server will return an LDAP_UNAVAILABLE_CRITICAL_EXTENSION error. - If 0, the control is not critical to the operation. Even if the server does not support the control, the operation is still attempted and the control is ignored.
ctrlp	Pointer to an LDAPControl structure created by this function. When you are done using this control, you should free it by calling ldap_control_free().

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if an invalid parameter was passed to the function.

- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the control.

Description

The `ldap_create_persistentsearch_control()` function allows you to perform persistent searches. A *persistent search* provides the means to track changes to a set of entries that match the search criteria. After the initial search is performed, the server keeps track of the search criteria and sends back information when any entry that matches the criteria is added, deleted, modified, or renamed. Calling `ldap_create_persistentsearch_control()` creates an LDAP server control that you can pass to the `ldap_search_ext()` function.

As stated, after you create the control, you can pass it to the LDAP server during a search operation. If you specify that you want “entry change notification” controls sent back (that is, if you specify a non-zero value for the `return_echg_ctls` parameter), the server includes controls with each changed entry it sends back. To retrieve the controls from each entry, call the `ldap_get_entry_controls()` function. To get data about the changes made to the entry from the control, call the `ldap_parse_entrychange_control()` function.

When you are done with the search, you can cancel the persistent search by calling the `ldap_abandon_ext()` function. You should also free the control from memory by calling the `ldap_control_free()` function.

See Also

`ldap_search_ext()`, `ldap_abandon_ext()`, `ldap_get_entry_controls()`, `ldap_parse_entrychange_control()`, `ldap_control_free()`

ldap_create_geteffectiveRights_control()

The `ldap_create_geteffectiveRights_control()` function allows a user (with correct privileges) to query about the access rights another identity has on one or more entries in the directory.

Syntax

```
#include <ldap-extension.h>
int ldap_create_geteffectiveRights_control( LDAP *ld, char *authzid,
    char *attrlist, char ctl_iscritical, LDAPControl **ctrlp);
```

Parameters

This function has the parameters listed in Table 17-63.

Table 17-63 ldap_create_geteffectiveRights_control() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
authzid	The DN of the identity for which you are checking access rights.
attrlist	A comma-separated list of access rights to determine.
ctl_iscritical	Specifies whether the control is critical to the operation. • If non-zero, the control is critical to the operation. If the server does not support it, the server will return an LDAP_UNAVAILABLE_CRITICAL_EXTENSION error. • If 0, the control is not critical to the operation. Even if the server does not support the control, the operation is still attempted and the control is ignored.
ctrlp	Pointer to an LDAPControl structure created by this function. When you are done using this control, you should free it by calling ldap_control_free().

Description

Typically, this control is used in an LDAP search request where the base, the scope, the filter and the attributes will determine the entries to return the access rights.

NOTE	This control is specific to Sun Java System Directory Server 5.2 and later.

ldap_create_proxiedauth_control()

The ldap_create_proxiedauth_control() function creates an LDAPv3 control that allows a bound entity to assume the identity of another directory entry.

NOTE	ldap_create_proxiedauth_control() is a new version of the ldap_create_proxyauth_control() function. If you are writing a new LDAP client, use ldap_create_proxiedauth_control().

Syntax

```
#include <ldap.h>
int ldap_create_proxiedauth_control( LDAP *ld, char *authzid,
    LDAPControl **ctrlp);
```

Parameters

This function has the parameters listed in Table 17-64.

Table 17-64 ldap_create_proxiedauth_control() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
authzid	The string representing the identity to assume for the access rights, defined in the format specified by the following Augmented Backus-Naur Form (ABNF) (copied from RFC 2829): <pre> ; Specific predefined authorization (authz) id schemes are ; defined below -- new schemes may be defined in the future. authzId = dnAuthzId / uAuthzId ; distinguished-name-based authz id. dnAuthzId = "dn:" dn dn = utf8string ; with syntax defined in RFC 2253 ; unspecified userid, UTF-8 encoded. uAuthzId = "u:" userid userid = utf8string ; syntax unspecified</pre>
ctrlp	Pointer to an LDAPControl structure created by this function. When you are done using this control, you should free it by calling ldap_control_free().

Description

More information on proxy authorization can be found in the *LDAP Proxied Authorization Control Internet Draft* (<http://ietfreport.isoc.org/ids/draft-weltman-ldapv3-proxy-12.txt>).

NOTE	Proxy authorization is an optional feature; it may not be supported on all LDAP servers. In order for the control to work, the server that you are connecting to must support the server control for proxy authorization (OID 2.16.840.1.113730.3.4.18, or LDAP_CONTROL_PROXIEDAUTH, as defined in the ldap-extension.h header file). See “Determining Supported Controls” on page 234 of Chapter 13, “Working with LDAP Controls”for information on determining the controls supported by a server.

ldap_create_proxyauth_control()

The ldap_create_proxyauth_control() function creates an LDAPv3 control that allows a bound entity to assume the identity of another directory entry.

NOTE This is an older function that is included in the SDK for backward-compatibility. If you are writing a new LDAP client, use `ldap_create_proxiedauth_control()`.

Syntax

```
#include <ldap.h>
int ldap_create_proxyauth_control( LDAP *ld, char *DN,
    char ctl_iscritical, LDAPControl **ctrlp);
```

Parameters

This function has the parameters listed in Table 17-65.

Table 17-65 ldap_create_proxyauth_control() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
DN	String representing the DN of the entry who's identity the client will be assuming.
ctl_iscritical	Specifies whether the control is critical to the operation. - If non-zero, the control is critical to the operation. If the server does not support it, the server will return an LDAP_UNAVAILABLE_CRITICAL_EXTENSION error. - If 0, the control is not critical to the operation. Even if the server does not support the control, the operation is still attempted and the control is ignored.
ctrlp	Pointer to an LDAPControl structure created by this function. When you are done using this control, you should free it by calling <code>ldap_control_free()</code> .

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if an invalid parameter was passed to the function.
- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the control.
- LDAP_UNAVAILABLE_CRITICAL_EXTENSION if the server does not support proxied authorization and `ctl_iscritical` is set to a non-zero value.

See Also
ldap_control_free()

ldap_create_sort_control()

The ldap_create_sort_control() function creates a control that specifies the order in which you want search results returned.

NOTE

This function implements an extension to the LDAPv3. Server-side sorting is an optional feature; it may not be supported on all LDAP servers. Call this function when interacting with LDAP servers that support this LDAPv3 extension. See “Determining Supported Extended Operations” on page 268 of Chapter 15, “Extended Operations” to determine if the server supports extended operations.

Syntax

```
#include <ldap.h>
int ldap_create_sort_control( LDAP *ld,
    LDAPsortkey **sortKeyList, const char ctl_iscritical,
    LDAPControl **ctrlp );
```

Parameters
This function has the parameters listed in Table 17-66.

Table 17-66 ldap_create_sort_control() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
sortKeyList	Pointer to an array of LDAPsortkey structures that specify the attribute types or matching rules used for sorting and the order (ascending or descending) in which to sort the results.
ctl_iscritical	Specifies whether the control is critical to the operation. • If non-zero, the control is critical to the operation. If the server does not support it, the server will return an LDAP_UNAVAILABLE_CRITICAL_EXTENSION error. • If 0, the control is not critical to the operation. Even if the server does not support the control, the operation is still attempted and the control is ignored.
ctrlp	Pointer to an LDAPControl structure created by this function. When you are done using this control, you should free it by calling ldap_control_free().

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if an invalid parameter was passed to the function.
- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the control.

Description

The `ldap_create_sort_control()` function allows you to specify the order in which you want to receive data from the server. Calling this function creates an LDAP control that you can pass to the `ldap_search_ext()` and `ldap_search_ext_s()` functions.

NOTE

In order for the control to work, the LDAP server that you are connecting to must support the server control for sorting search results (OID 1.2.840.113556.1.4.473, or LDAP_CONTROL_SORTREQUEST, as defined in the `ldap.h` header file). See “Determining Supported Controls” on page 234 of Chapter 13, “Working with LDAP Controls” for information on determining the controls supported by a server.

To specify the attributes to use for sorting the results, you can call `ldap_create_sort_keylist()` to create an array of `LDAPsortkey` structures and pass the array as the `sortKeyList` argument. When you are done with the search, you should free the control and the array of `LDAPsortkey` structures by calling the `ldap_control_free()` and `ldap_free_sort_keylist()` functions.

For more information, see “Creating the Control” on page 238 of Chapter 13, “Working with LDAP Controls.”

See Also

`ldap_create_sort_keylist()`, `ldap_search_ext()`, `ldap_search_ext_s()`, `ldap_control_free()`

ldap_create_sort_keylist()

The `ldap_create_sort_keylist()` function creates an array of `LDAPsortkey` structures from a string representation of a set of sort keys.

Syntax

```
#include <ldap.h>
int ldap_create_sort_keylist(LDAPsortkey ***sortKeyList,
    const char *string_rep);
```

Parameters

This function has the parameters listed in Table 17-67.

Table 17-67 ldap_create_sort_keylist() Function Parameters

Parameter	Description
sortKeyList	Pointer to an array of LDAPsortkey structures that specify the attribute types or matching rules used for sorting and the order (ascending or descending) in which to sort the results.
string_rep	String representation of a set of sort keys. The value should specify the name of the attribute that you want to sort by. To sort in reverse order, precede the attribute name with a hyphen ("-"). To use a matching rule for sorting, append a colon to the attribute name and specify the object identifier (OID) of a matching rule after the colon. For example: 1. cn (sorts by the cn attribute) 2. -cn (sorts by the cn attribute in reverse order) 3. -cn:1.2.3.4 (sorts by the cn attribute in reverse order using the matching rule identified by the OID 1.2.3.4)

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if an invalid parameter was passed to the function.
- LDAP_NO_MEMORY if memory cannot be allocated.
- -1 if an error occurred.

Description

Calling the ldap_create_sort_keylist() function allows you to create an array of LDAPsortkey structures from a string representation of a set of sort keys. You can then pass the array to the ldap_create_sort_control() function. The string representation specified by the string_rep argument should specify the name of the attribute that you want to sort by and how the sort will work. When you are done sorting the results, you should free the array of LDAPsortkey structures by calling the ldap_free_sort_keylist() function.

For more information, see “Specifying the Sort Order” on page 237 of Chapter 13, “Working with LDAP Controls.”

See Also

ldap_create_sort_control(), ldap_free_sort_keylist()

ldap_create_virtuallist_control()

The ldap_create_virtuallist_control() function creates a control that requests a subset of search results for use in a virtual list box.

NOTE	ldap_create_virtuallist_control() implements an extension to the LDAPv3 supported by Sun Java System Directory Server. For information on determining if another server supports this control, see “Determining Supported Controls” on page 234 of Chapter 13, “Working with LDAP Controls.”

Syntax

```
#include <ldap.h>
int ldap_create_virtuallist_control( LDAP *ld,
    LDAPVirtualList *ldvlistp, LDAPControl **ctrlp );
```

Parameters

This function has the parameters listed in Table 17-68.

Table 17-68 ldap_create_virtuallist_control() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
ldvlistp	Pointer to an LDAPVirtualList structure that specifies the subset of entries that you want retrieved from the server and the selected entry.
ctrlp	Pointer to an LDAPControl structure created by this function. When you are done using this control, you should free it by calling ldap_control_free().

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if an invalid parameter was passed to the function.

- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the control.

Description

The `ldap_create_virtuallist_control()` function allows you to retrieve a subset of entries from the server for use in a virtual list box. To specify the subset of entries that you want to retrieve, create an `LDAPVirtualList` structure and pass in a pointer to this structure as the `ldvlistp` argument.

NOTE

In order for the control to work, the LDAP server that you are connecting to must support the server control for sorting search results (OID 2.16.840.1.113730.3.4.9, or `LDAP_CONTROL_VLVREQUEST`, as defined in the `ldap.h` header file). See “Determining Supported Controls” on page 234 of Chapter 13, “Working with LDAP Controls” for information on determining the controls supported by a server.

Calling `ldap_create_virtuallist_control()` creates an LDAP control that you can pass to the `ldap_search_ext()` and `ldap_search_ext_s()` functions. You also need to pass a server-side sorting control to the search functions. You can call `ldap_create_sort_keylist()` and `ldap_create_sort_control()` to create a server-side sorting control. When you are done with the search, you should free the control by calling the `ldap_control_free()` function.

For more information about this control, see “Using the Virtual List View Control” on page 245 of Chapter 13, “Working with LDAP Controls.”

See Also

`LDAPVirtualList`, `ldap_parse_virtuallist_control()`, `ldap_search_ext()`, `ldap_search_ext_s()`, `ldap_control_free()`

ldap_delete()

The `ldap_delete()` function removes an entry from the directory asynchronously.

NOTE

This is an older function that is included in the SDK for backward-compatibility. If you are writing a new LDAP client, use `ldap_delete_ext()` instead.

Syntax

```
#include <ldap.h>
int ldap_delete( LDAP *ld, const char *dn );
```


Parameters

This function has the parameters listed in Table 17-69.

Table 17-69 ldap_delete() FunctionParameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
dn	DN of the entry to remove.

Returns

Returns the message ID of the ldap_delete() operation.

NOTE To check the result of this operation, call ldap_result() and ldap_result2error(). See ldap_delete_ext_s() for a list of possible result codes for the LDAP delete operation.

Description

Please use the newer version of this function, ldap_delete_ext().

Example

Code Example 17-11 uses the asynchronous ldap_delete() function to remove the entry for *Barbara Jensen* from the directory.

Code Example 17-11 ldap_delete() Code Example

```
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result;
int msgid, rc;
struct timeval tv;

/* Distinguished name of the entry that you want to delete. */
char *dn = "uid=bjensen,ou=People,dc=example, dc=com";
...
/* Set up the timeout period to wait for the "modify" operation */
tv.tv_sec = tv.tv_usec = 0;

/* Delete the entry */
if ( ( msgid = ldap_delete( ld, dn ) ) == -1 ) {
    ldap_perror( ld, "ldap_delete" );
    return( 1 );
}
```

Code Example 17-11 ldap_delete() Code Example (*Continued*)

```
/* Check to see if the operation has completed */
while ( ( rc = ldap_result( ld, msgid, 0, &tv, &result ) ) == 0 ) {
    ...
    /* do other work while waiting for the operation to complete */
    ...
}
/* Check the result to see if any errors occurred */
ldap_result2error( ld, result, 1 );
ldap_perror( ld, "ldap_delete" );
...
```

See Also

ldap_delete_ext()

ldap_delete_ext()

The ldap_delete_ext() function deletes an entry from the directory asynchronously.

NOTE ldap_delete_ext() is a new version of the ldap_delete() function. If you are writing a new LDAP client, you should call ldap_delete_ext().

Syntax

```
#include <ldap.h>
int ldap_delete_ext( LDAP *ld, const char *dn,
    LDAPControl **serverctrls, LDAPControl **clientctrls, int *msgidp );
```

Parameters

This function has the parameters listed in Table 17-70.

Table 17-70 ldap_delete_ext() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
dn	DN of the entry to remove.
serverctrls	Pointer to an array of LDAPControl structures representing LDAP server controls that apply to this operation. If you do not want to pass any server controls, specify NULL for this argument.

Table 17-70 ldap_delete_ext() Function Parameters (*Continued*)

Parameter	Description
clientctrls	Pointer to an array of LDAPControl structures representing LDAP client controls that apply to this operation. If you do not want to pass any client controls, specify NULL for this argument.
msgidp	Pointer to an integer that will be set to the message ID of the LDAP operation.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if an invalid parameter was passed to the function.
- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the control.
- LDAP_SERVER_DOWN if the LDAP server did not receive the request or if the connection to the server was lost.
- LDAP_NOT_SUPPORTED if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3. Make sure that you set the version of your LDAP client to version 3 before calling this function. For details, see “Specifying the LDAP Version of Your Client” on page 65 of Chapter 3, “Writing an LDAP Client.”

NOTE

To check the result of this operation, call ldap_result() and ldap_result2error(). See ldap_delete_ext_s() for a list of possible result codes for the LDAP delete operation.

Description

The ldap_delete_ext() function deletes an entry from the directory, using the dn argument to specify the entry that you want to delete. ldap_delete_ext() is an asynchronous function; it does not directly return results. If you want the results to be returned directly by the function, call the synchronous function ldap_delete_ext_s().

For additional information on deleting entries from the directory, see Chapter 7, “Adding, Modifying and Deleting Entries.”

Example

See Code Example 4-2 on page 77 of Chapter 4, “Using the LDAP API” and Code Example 7-21 on page 177 of Chapter 7, “Adding, Modifying and Deleting Entries.”

See Also

ldap_delete_ext_s(), ldap_result(), ldap_parse_result()

ldap_delete_ext_s()

The ldap_delete_ext_s() function deletes an entry from the directory synchronously.

NOTE ldap_delete_ext_s() is a new version of the ldap_delete_s() function. If you are writing a new LDAP client, you should call ldap_delete_ext_s().

Syntax

```
#include <ldap.h>
int ldap_delete_ext_s( LDAP *ld, const char *dn,
    LDAPControl **serverctrls, LDAPControl **clientctrls );
```

Parameters

This function has the parameters listed in Table 17-71.

Table 17-71 ldap_delete_ext_s() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
dn	DN of the entry to remove.
serverctrls	Pointer to an array of LDAPControl structures representing LDAP server controls that apply to this operation. If you do not want to pass any server controls, specify NULL for this argument.
clientctrls	Pointer to an array of LDAPControl structures representing LDAP client controls that apply to this operation. If you do not want to pass any client controls, specify NULL for this argument.

Returns

One of the following values:

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if an invalid parameter was passed to the function.
- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the control.
- LDAP_SERVER_DOWN if the LDAP server did not receive the request or if the connection to the server was lost.
- LDAP_NOT_SUPPORTED if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3. Make sure that you set the version of your LDAP client to version 3 before calling this function. For details, see “Specifying the LDAP Version of Your Client” on page 65 of Chapter 3, “Writing an LDAP Client.”
- LDAP_LOCAL_ERROR if an error occurred when receiving the results from the server.
- LDAP_DECODING_ERROR if an error occurred when decoding the BER-encoded results from the server.

NOTE

Sun Java System Directory Server and other LDAP server products may send result codes in addition to those described. For example, the server may have loaded a plug-in that returns custom result codes. Check your LDAP server documentation for other result codes.

Description

The `ldap_delete_ext_s()` function deletes an entry from the directory, using the `dn` argument to specify the entry that you want to delete. `ldap_delete_ext_s()` is a synchronous function, which directly returns the results of the operation. If you want to perform other operations while waiting for the results of this operation, call the asynchronous function `ldap_delete_ext()` instead.

For additional information on deleting entries from the directory, see Chapter 7, “Adding, Modifying and Deleting Entries.”

Example

See Code Example 4-1 on page 76 of Chapter 4, “Using the LDAP API” and Code Example 7-19 on page 175 in Chapter 7, “Adding, Modifying and Deleting Entries.”

See Also

ldap_delete_ext()

ldap_delete_s()

The `ldap_delete_s()` deletes an entry from the directory synchronously.

NOTE

This is an older function that is included in the SDK for backward-compatibility. If you are writing a new LDAP client, use `ldap_delete_ext_s()` instead.

Syntax

```
#include <ldap.h>
int ldap_delete_s(LDAP *ld, const char *dn);
```

Parameters

This function has the parameters listed in Table 17-72.

Table 17-72 ldap_delete_s() Function Parameters

Parameters	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
dn	DN of the entry to remove.

Returns

For a list of possible result codes for an LDAP delete operation, see the `ldap_delete_ext_s()` function.

Description

Please use the newer version of this function, `ldap_delete_ext_s()`.

Example

Code Example 17-12 uses the synchronous `ldap_delete_s()` function to delete the entry for *Barbara Jensen* from the directory.

Code Example 17-12 ldap_delete_s() Code Example

```

#include <ldap.h>
LDAP *ld;

/* Distinguished name of the entry that you want to delete. */
char *dn = "uid=bjensen,ou=People,dc=example, dc=com";
...
/* Delete the entry */
if ( ldap_delete_s( ld, dn ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_delete_s" );
    return( 1 );
}
...

```

See Also

ldap_delete_ext_s()

ldap_dn2ufn()

The `ldap_dn2ufn()` function converts a DN into a more user-friendly form by stripping off the cryptic type names.

Syntax

```

#include <ldap.h>
char * ldap_dn2ufn( const char *dn );

```

Parameters

This function has the parameter listed in Table 17-73.

Table 17-73 ldap_dn2ufn() Function Parameter

Parameter	Description
dn	The DN that you want converted.

Returns

One of the following values:

- If successful, returns the DN in its friendlier form.
- If unsuccessful, returns NULL.

Description

For more information on ldap_dn2ufn(), see the man page. For more information on user friendly naming (UFN), see RFC 1781, *Using the OSI Directory to Achieve User Friendly Naming* (<http://www.faqs.org/rfcs/rfc1781.html>).

ldap_entry2html()

The ldap_entry2html() function writes the HTML representation of an LDAP entry.

Syntax

```
#include <disptmpl.h>
int ldap_entry2html( LDAP *ld, char *buf, LDAPMessage *entry,
    struct ldap_disptmpl *tmpl, char **defattrs, char ***defvals,
    writeproc writeproc, void *writeparm, char *eol, int rdncount,
    unsigned long opts, char *urlprefix, char *base );
```

Parameters

This function has the parameters listed in Table 17-74.

Table 17-74 ldap_entry2html() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
buf	A pointer to a buffer of size LDAP_DTmpl_BUFSIZ or larger. If NULL, a buffer is allocated and freed internally.
entry	Specifies the attribute values to be represented.
tmpl	Pointer to the display template to be used, usually obtained by calling ldap_oc2template().
defattrs	A NULL-terminated array of LDAP attribute names for which you wish to provide default values. It is only used if the entry contains no values for the attribute.
defvals	An array of NULL-terminated arrays of default values corresponding to the attributes.

Table 17-74 ldap_entry2html() Function Parameters (*Continued*)

Parameter	Description
writeproc	writeproc function should be declared as: <pre>int writeproc(writeparm, p, len) void *writeparm; char *p; int len;</pre> where p is a pointer to text to be written and len is the length of the text. p is guaranteed to be zero-terminated.
writeparm	A pointer to a structure that will be passed as the first parameter of the writeproc procedure. Typically, this is used to pass the file descriptor of the file to write to.
eol	Lines of text are terminated with this string.
rdncount	Limits the number of components that are displayed for DN attributes.
opts	Specifies output options. The allowed values are: • zero (default output) • LDAP_DISP_OPT_AUTOLABELWIDTH which causes the width for labels to be determined based on the longest label in tmp1. • LDAP_DISP_OPT_HTMLBODYONLY which instructs the library not to include <HTML>, <HEAD>, <TITLE>, and <BODY> tags. In other words, an HTML fragment is generated, and the caller is responsible for prepending and appending the appropriate HTML tags to construct a correct HTML document.
urlprefix	Starting text to use when constructing an LDAP URL. The default is the string ldap://.
base	The base with which to begin when executing search actions. If NULL, search action template items are ignored.

Returns

- LDAP_SUCCESS if successful.
- An LDAP error code on error.

Description

ldap_entry2html() produces an HTML representation of an entry. It behaves exactly like ldap_entry2text() except for the output and the addition of two parameters.

See Also

ldap_entry2text(), ldap_oc2template()

ldap_entry2html_search()

The ldap_entry2html_search() function determines the appropriate display template to use by calling ldap_oc2template().

Syntax

```
#include <disptmpl.h>
int ldap_entry2html_search( LDAP *ld, char *dn, char *base,
    LDAPMessage *entry, struct ldap_disptmpl *tmplist, char **defattrs,
    char ***defvals, writeproc writeproc, void *writeparm, char *eol,
    int rdncount, unsigned long opts, char *urlprefix );
```

Parameters

This function has the parameters listed in Table 17-75.

Table 17-75 ldap_entry2html_search() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
dn	DN of the entry to write as HTML.
base	The base with which to begin when executing search actions. If NULL, search action template items are ignored.
entry	Specifies the attribute values to be represented.
tmplist	Pointer to the entire list of templates available (usually obtained by calling ldap_init_templates() or ldap_init_templates_buf()). If NULL, will attempt to read a load templates from the default template configuration file.
defattrs	A NULL-terminated array of LDAP attribute names for which you wish to provide default values. It is only used if the entry contains no values for the attribute.
defvals	An array of NULL-terminated arrays of default values corresponding to the attributes.
writeproc	writeproc function should be declared as: int writeproc(writeparm, p, len) void *writeparm; char *p; int len; where p is a pointer to text to be written and len is the length of the text. p is guaranteed to be zero-terminated.
writeparm	A pointer to a structure that will be passed as the first parameter of the writeproc procedure. Typically, this is used to pass the file descriptor of the file to write to.
eol	Lines of text are terminated with this string.

Table 17-75 ldap_entry2html_search() Function Parameters (*Continued*)

Parameter	Description
rdncount	Limits the number of components that are displayed for DN attributes.
opts	Specifies output options. The allowed values are: • zero (default output) • LDAP_DISP_OPT_AUTOLABELWIDTH which causes the width for labels to be determined based on the longest label in tmpl. • LDAP_DISP_OPT_HTMLBODYONLY which instructs the library not to include <HTML>, <HEAD>, <TITLE>, and <BODY> tags. In other words, an HTML fragment is generated, and the caller is responsible for prepending and appending the appropriate HTML tags to construct a correct HTML document.
urlprefix	Starting text to use when constructing an LDAP URL. The default is the string ldap://

Returns

- LDAP_SUCCESS if successful.
- An LDAP error code on error.

Description

ldap_entry2html_search() will call ldap_search_s() to retrieve any attribute values to be displayed. It behaves exactly like ldap_entry2text_search(), except the urlprefix parameter is required and HTML is the output.

See Also

ldap_search_s(), ldap_entry2text_search(), ldap_init_templates(), ldap_init_templates_buf(), ldap_entry2text()

ldap_entry2text()

The ldap_entry2text() function writes the text representation of an LDAP entry.

Syntax

```
#include <disptmpl.h>
int ldap_entry2text( LDAP *ld, char *buf, LDAPMessage *entry,
    struct ldap_disptmpl *tmpl, char **defattrs, char ***defvals,
    writetype writeproc, void *writeparm, char *eol, int rdncount,
    unsigned long opts );
```

Parameters

This function has the parameters listed in Table 17-76.

Table 17-76 ldap_entry2text() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
buf	A pointer to a buffer of size LDAP_DTML_BUFSIZ or larger. If NULL, a buffer is allocated and freed internally.
entry	Specifies the attribute values to be represented.
tmpl	Pointer to the display template to be used, usually obtained by calling ldap_oc2template().
defattrs	A NULL-terminated array of LDAP attribute names for which you wish to provide default values. It is only used if the entry contains no values for the attribute.
defvals	An array of NULL-terminated arrays of default values corresponding to the attributes.
writeproc	writeproc function should be declared as: <pre>int writeproc(writeparm, p, len) void *writeparm; char *p; int len;</pre> where p is a pointer to text to be written and len is the length of the text. p is guaranteed to be zero-terminated.
writeparm	A pointer to a structure that will be passed as the first parameter of the writeproc procedure. Typically, this is used to pass the file descriptor of the file to write to.
eol	Lines of text are terminated with this string.
rdncount	Limits the number of components that are displayed for DN attributes.
opts	Specifies output options. The allowed values are: • zero (default output) • LDAP_DISP_OPT_AUTOLABELWIDTH which causes the width for labels to be determined based on the longest label in tmpl. • LDAP_DISP_OPT_HTMLBODYONLY which instructs the library not to include <HTML>, <HEAD>, <TITLE>, and <BODY> tags. In other words, an HTML fragment is generated, and the caller is responsible for prepending and appending the appropriate HTML tags to construct a correct HTML document.

Returns

- LDAP_SUCCESS if successful.
- An LDAP error code on error.

Description

ldap_entry2text() produces a text representation of an entry and writes the text by calling the writeproc function. All of the attribute values to be displayed must be present in entry as no interaction with the LDAP server will be performed.

See Also

ldap_entry2html(), ldap_oc2template()

ldap_entry2text_search()

The ldap_entry2text_search() function determines the appropriate display template to use by calling ldap_oc2template().

Syntax

```
#include <disptmpl.h>
int ldap_entry2text_search( LDAP *ld, char *dn, char *base,
    LDAPMessage *entry, struct ldap_disptmpl *tmplist, char **defattrs,
    char ***defvals, writeptype writeproc, void *writeparm,
    char *eol, int rdncount, unsigned long opts );
```

Parameters

This function has the parameters listed in Table 17-77.

Table 17-77 ldap_entry2text_search() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
dn	DN of the entry to write.
base	If NULL, the search action template items are ignored. If not NULL, it is the search base to use when executing search actions.
entry	If entry is not NULL, it should contain the objectClass attribute values for the entry to be displayed. If NULL, dn can not be NULL, and ldap_entry2text_search() will retrieve the objectClass values itself by calling ldap_search_s().
tmplist	Pointer to the entire list of templates available (usually obtained by calling ldap_init_templates() or ldap_init_templates_buf()). If NULL, will attempt to read a load templates from the default template configuration file.

Table 17-77 ldap_entry2text_search() Function Parameters (*Continued*)

Parameter	Description
defattr	A NULL-terminated array of LDAP attribute names for which you wish to provide default values. It is only used if the entry contains no values for the attribute.
defvals	An array of NULL-terminated arrays of default values corresponding to the attributes.
writeproc	writeproc function should be declared as: <pre>int writeproc(writeparm, p, len) void *writeparm; char *p; int len;</pre> where p is a pointer to text to be written and len is the length of the text. p is guaranteed to be zero-terminated.
writeparm	A pointer to a structure that will be passed as the first parameter of the writeproc procedure. Typically, this is used to pass the file descriptor of the file to write to.
eol	Lines of text are terminated with this string.
rdncount	Limits the number of components that are displayed for DN attributes.
opts	Specifies output options. The allowed values are: • zero (default output) • LDAP_DISP_OPT_AUTOLABELWIDTH which causes the width for labels to be determined based on the longest label in tmp1. • LDAP_DISP_OPT_HTMLBODYONLY which instructs the library not to include <HTML>, <HEAD>, <TITLE>, and <BODY> tags. In other words, an HTML fragment is generated, and the caller is responsible for prepending and appending the appropriate HTML tags to construct a correct HTML document.

Returns

- LDAP_SUCCESS if successful.
- An LDAP error code on error.

Description

ldap_entry2text_search() will call ldap_search_s() to retrieve any attribute values to be displayed.

See Also

ldap_entry2html_search(), ldap_oc2template(), ldap_search_s(),
ldap_init_templates(), ldap_init_templates_buf()

ldap_err2string()

The `ldap_err2string()` function returns the corresponding error message for an error code.

Syntax

```
#include <ldap.h>
char * ldap_err2string( int err );
```

Parameters

This function has the parameter listed in Table 17-78.

Table 17-78 ldap_err2string() Function Parameter

Parameter	Description
err	Error code that you want interpreted into an error message.

Returns

One of the following values:

- If successful, returns the corresponding error message for the error code.
- If unsuccessful (for example, if the error code is not a valid LDAP API error code), returns `Unknown error`.

Description

For more information, see “Getting the Error Message from an Error Code” on page 91 of Chapter 4, “Using the LDAP API.”

Example

Code Example 17-13 sets the variable `err_msg` to the error message corresponding to the error code returned by the `ldap_simple_bind_s()` function.

Code Example 17-13 ldap_err2string() Code Example

```
#include <ldap.h>
...
LDAP *ld;
char *dn = "uid=bjensen,ou=People,dc=example, dc=com";
char *pw = "hifalutin";
char *err_msg;
...
```

Code Example 17-13 ldap_err2string() Code Example

```
err_msg = ldap_err2string( ldap_simple_bind_s( ld, dn, pw ) );
...
```

See Also

ldap_get_lderrno(), ldap_perror(), ldap_result2error(),
ldap_set_lderrno(), ldapssl_err2string()

ldap_explode_dn()

The ldap_explode_dn() function converts a DN into its component parts.

Syntax

```
#include <ldap.h>
char ** ldap_explode_dn( const char *dn, int notypes );
```

Parameters

This function has the parameters listed in Table 17-79.

Table 17-79 ldap_explode_dn() Function Parameters

Parameter	Description
dn	DN that you want separated into components.
notypes	Specifies whether or not type names in the DN are returned. This parameter can have the following possible values: • 0 specifies that type names are returned. • A non-zero value specifies that type names are not returned.

Returns

One of the following values:

- If successful, returns a NULL-terminated array containing the components of the DN.
- If unsuccessful, returns NULL.

Description

For more information, see “Getting the Components of a Distinguished Name” on page 126 of Chapter 5, “Searching a LDAP Directory.”

Example

The following function call:

```
ldap_explode_dn( "uid=bjensen,ou=People,dc=example,dc=com", 0 )
```

returns this array:

```
{ "uid=bjensen", "ou=People", "dc=example,dc=com", NULL }
```

If you change the `otypes` parameter from 0 to 1:

```
ldap_explode_dn( "uid=bjensen,ou=People,dc=example,dc=com", 1 )
```

The component names are not returned in the array:

```
{ "bjensen", "People", "example.com", NULL }
```

See Also

`ldap_explode_rdn()`, `ldap_get_dn()`

ldap_explode_dns()

CAUTION This function is to be deprecated and should not be used. It is included in `ldap-to-be-deprecated.h` for backward-compatibility.

The `ldap_explode_dns()` function takes a DNS-style DN, breaks it up into its component parts, and returns a NULL-terminated array. For example, the DN `cal.sun.com` will return `{ "cal", "sun", "com", NULL }`. The result can be freed by calling `ldap_value_free()`.

Syntax

```
#include <ldap-to-be-deprecated.h>
char ** ldap_explode_dns( const char *dn );
```

Parameters

This function has the parameter listed in Table 17-80.

Table 17-80 ldap_explode_dns() Function Parameter

Parameter	Description
dn	DN that you want separated into components.

Returns

One of the following values:

- If successful, returns a NULL-terminated array containing the components of the DN.
- If unsuccessful, returns NULL.

Description

For more information, see “Getting the Components of a Distinguished Name” on page 126 of Chapter 5, “Searching a LDAP Directory.”

ldap_explode_rdn()

The `ldap_explode_rdn()` function converts a relative distinguished name (RDN) into its component parts.

Syntax

```
#include <ldap.h>
char ** ldap_explode_rdn( const char *dn, int notypes );
```

Parameters

This function has the parameters listed in Table 17-81.

Table 17-81 ldap_explode_rdn() Function Parameters

Parameter	Description
dn	RDN that you want separated into components.
notypes	Specifies whether or not type names in the RDN are returned. This parameter can have the following possible values: • 0 specifies that type names are returned. • A non-zero value specifies that type names are not returned.

Returns

One of the following values:

- If successful, returns a NULL-terminated array containing the components of the RDN.
- If unsuccessful, returns NULL.

Description

For more information, see “Getting the Components of a Distinguished Name” on page 126 of Chapter 5, “Searching a LDAP Directory.”

Example

The following function call:

```
ldap_explode_rdn( "ou=Sales + cn=Barbara Jensen", 0 );
```

returns this array:

```
{ "ou=Sales", "cn=Barbara Jensen", NULL }
```

See Also

ldap_explode_dn(), ldap_get_dn()

ldap_extended_operation()

The `ldap_extended_operation()` function sends a request to the server to perform an extended operation asynchronously.

Syntax

```
#include <ldap.h>
int ldap_extended_operation( LDAP *ld, const char *requestoid,
    struct berval *requestdata, LDAPControl **serverctrls,
    LDAPControl **clientctrls, int *msgidp );
```

Parameters

This function has the parameters listed in Table 17-82.

Table 17-82 ldap_extended_operation() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.

Table 17-82 ldap_extended_operation() Function Parameters *(Continued)*

Parameter	Description
requestoid	Object identifier (OID) of the extended operation that you want the server to perform. (After processing an LDAPv3 extended operation, an LDAP server can return an OID and data in the result. To parse the OID and data from the result, call the ldap_parse_extended_result() function.)
requestdata	Pointer to a berval structure containing the data that you want passed to the server to perform the extended operation. The data in the berval is a buffer of the BER encoded datatype, usually obtained using the ber_flatten() function.
serverctrls	Pointer to an array of LDAPControl structures representing LDAP server controls that apply to this operation. If you do not want to pass any server controls, specify NULL for this argument.
clientctrls	Pointer to an array of LDAPControl structures representing LDAP client controls that apply to this operation. If you do not want to pass any client controls, specify NULL for this argument.
msgidp	Pointer to an integer that will be set to the message ID of the LDAP operation.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if an invalid parameter was passed to the function.
- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the control.
- LDAP_SERVER_DOWN if the LDAP server did not receive the request or if the connection to the server was lost.
- LDAP_NOT_SUPPORTED if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3. Make sure that you set the version of your LDAP client to version 3 before calling this function. For details, see “Specifying the LDAP Version of Your Client” on page 65 of Chapter 3, “Writing an LDAP Client.”

NOTE	To check the result of this operation, call ldap_result(), ldap_parse_extended_result(), and ldap_get_lderrno().

Description

The `ldap_extended_operation()` function sends a request to the server to perform an LDAPv3 extended operation. `ldap_extended_operation()` is an asynchronous function; it does not directly return results. If you want the results to be returned directly by the function, call the synchronous function `ldap_extended_operation_s()`.

For additional information on asynchronous extended operations, see “Asynchronous Extended Operation” on page 268 of Chapter 15, “Extended Operations.” For a list of possible result codes for an LDAP extended operation, see `ldap_extended_operation_s()`.

See Also

`ldap_extended_operation_s()`, `ldap_result()`, `ldap_parse_extended_result()`, `ldap_get_lderrno()`, `ber_flatten()`

ldap_extended_operation_s()

The `ldap_extended_operation_s()` function sends a request to the server to perform an extended operation synchronously.

Syntax

```
#include <ldap.h>
int ldap_extended_operation_s( LDAP *ld, const char *requestoid,
    struct berval *requestdata, LDAPControl **serverctrls,
    LDAPControl **clientctrls, char **retoidp, struct berval **retdatap );
```

Parameters

This function has the parameters listed in Table 17-83.

Table 17-83 `ldap_extended_operation_s()` Function Parameters

Parameter	Description
<code>ld</code>	Pointer to an LDAP structure containing information about the connection to the LDAP server.
<code>requestoid</code>	Object identifier (OID) of the extended operation that you want the server to perform.
<code>requestdata</code>	Pointer to a <code>berval</code> structure containing the data that you want passed to the server to perform the extended operation. The data in the <code>berval</code> is a buffer of the BER encoded datatype, usually obtained using the <code>ber_flatten()</code> function.

Table 17-83 ldap_extended_operation_s() Function Parameters (*Continued*)

Parameter	Description
serverctrls	Pointer to an array of LDAPControl structures representing LDAP server controls that apply to this operation. If you do not want to pass any server controls, specify NULL for this argument.
clientctrls	Pointer to an array of LDAPControl structures representing LDAP client controls that apply to this operation. If you do not want to pass any client controls, specify NULL for this argument.
retoidp	Pointer to the OID returned by the server after performing the extended operation. When done, you can free this by calling the ldap_memfree() function.
retdatap	Pointer to the pointer for a berval structure containing the data returned by the server after performing the extended operation. When done, you can free this by calling the ber_bvfree() function.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if an invalid parameter was passed to the function.
- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the control.
- LDAP_SERVER_DOWN if the LDAP server did not receive the request or if the connection to the server was lost.
- LDAP_LOCAL_ERROR if an error occurred when receiving the results from the server.
- LDAP_DECODING_ERROR if an error occurred when decoding the BER-encoded results from the server.
- LDAP_NOT_SUPPORTED if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3. Make sure that you set the version of your LDAP client to version 3 before calling this function. For details, see “Specifying the LDAP Version of Your Client” on page 65 of Chapter 3, “Writing an LDAP Client.”

NOTE

Sun Java System Directory Server and other LDAP server products may send result codes in addition to those described. For example, the server may have loaded a plug-in that returns custom result codes. Check your LDAP server documentation for other result codes.

Description

The `ldap_extended_operation_s()` function sends a request to the server to perform an LDAPv3 extended operation.

NOTE

The LDAP server must support the extended operation. Sun Java System Directory Server supports a server plug-in interface that you can use to add support for extended operations. For details, see the *Sun ONE Directory Server Plug-In Programming Guide*.

`ldap_extended_operation_s()` is a synchronous function, which directly returns the results of the operation. If you want to perform other operations while waiting for the results of this operation, call the asynchronous function `ldap_extended_operation()`. After processing an LDAPv3 extended operation, an LDAP server can return an object identifier and data in the results. The `retoidp` and `retdatap` arguments point to these values.

For information on determining the extended operations supported by a server, see “Determining Supported Extended Operations” on page 268 of Chapter 15, “Extended Operations.”

See Also

`ldap_extended_operation()`, `ber_flatten()`

ldap_first_attribute()

The `ldap_first_attribute()` function returns the name of the first attribute in an entry returned by the `ldap_first_entry()`, the `ldap_next_entry()`, or the `ldap_result()` functions.

Syntax

```
#include <ldap.h>
char * ldap_first_attribute( LDAP *ld, LDAPMessage *entry,
    BerElement **ber );
```

Parameters

This function has the parameters listed in Table 17-84.

Table 17-84 ldap_first_attribute() Function Parameters

Parameter	Description
<code>ld</code>	Pointer to an LDAP structure containing information about the connection to the LDAP server.

Table 17-84 ldap_first_attribute() Function Parameters

Parameter	Description
entry	Pointer to the LDAPMessage structure representing the entry returned by the ldap_first_entry() or ldap_next_entry() function.
ber	Pointer to a BerElement allocated to keep track of its current position. Pass this pointer to subsequent calls to ldap_next_attribute() to step through the entry's attributes.

Returns

One of the following values:

- If successful, returns the pointer to the name of the first attribute in an entry. When you are done using this data, you should free the memory by calling the ldap_memfree() function.
- If unsuccessful, returns a NULL and sets the appropriate error code in the LDAP structure. To get the error code, call the ldap_get_lderrno() function.

See Chapter 18, “Result Codes” for a complete listing of error codes.

Description

For more information, see “Getting Attribute Names from an Entry” on page 127 of Chapter 5, “Searching a LDAP Directory.”

Example

Code Example 17-14 retrieves each attribute for an entry.

Code Example 17-14 ldap_first_attribute() Code Example

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result, *e;
BerElement *ber;
char *a;
char *my_searchbase = "dc=example,dc=com";
char *my_filter = "(sn=Jensen)"
...
/* Search the directory */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, \
    my_filter, NULL, 0, &result ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_search_s" );
    return( 1 );
}
```


Code Example 17-14 ldap_first_attribute() Code Example (*Continued*)

```

/* Get the first matching entry.*/
e = ldap_first_entry( ld, result );

/* Retrieve the attributes the entry */
for ( a = ldap_first_attribute( ld, e, &ber ); a != NULL;
      a = ldap_next_attribute( ld, e, ber ) ) {
    ...
    /* Code to get and manipulate attribute values */
    ...
}
ldap_memfree( a );
}
/* Free the BerElement from memory when done */
if ( ber != NULL ) {
    ldap_ber_free( ber, 0 );
}
...

```

See Also

ldap_first_entry(), ldap_next_entry(), ldap_next_attribute(), BerElement

ldap_first_disptmpl()

The ldap_first_disptmpl() function returns the first template in a list.

Syntax

```

#include <disptmpl.h>
ldap_first_disptmpl( struct ldap_disptmpl *tmplist );

```

Parameters

This function has the parameter listed in Table 17-85.

Table 17-85 ldap_first_disptmpl() Function Parameter

Parameter	Description
tmplistp	Pointer to a list of template data structures.

Description

ldap_first_disptmpl() returns the first template in the list of templates pointed to by the parameter tmplist. tmplist is typically obtained by calling ldap_init_templates().

See Also

ldap_next_disptmpl(), ldap_init_templates(), ldap_init_templates_buf()

ldap_first_entry()

The ldap_first_entry() function returns a pointer to the LDAPMessage structure representing the first directory entry in a chain of search results.

Syntax

```
#include <ldap.h>
LDAPMessage * ldap_first_entry( LDAP *ld, LDAPMessage *result );
```

Parameters

This function has the parameters listed in Table 17-86.

Table 17-86 ldap_first_entry() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
result	Chain of search results, which are represented by a pointer to the LDAPMessage structure.

Returns

One of the following values:

- If successful, returns the pointer to the first LDAPMessage structure of the type LDAP_RES_SEARCH_ENTRY in the chain of search results.
- If no LDAPMessage structures of the type LDAP_RES_SEARCH_ENTRY are in the chain of the search results or if the function is unsuccessful, returns a NULLMSG.

Description

The `ldap_first_entry()` function returns a pointer to the `LDAPMessage` structure representing the first directory entry in a chain of search results. Search result entries are in messages of the type `LDAP_RES_SEARCH_ENTRY`. You can use this function in conjunction with the `ldap_next_entry()` function to iterate through the directory entries in a chain of search results. These functions skip over any messages in the chain that do not have the type `LDAP_RES_SEARCH_ENTRY`.

CAUTION Do not free the `LDAPMessage` structure returned by this function. Because this is a structure within a chain of search results, freeing this structure will free part of the chain of search results. When you are done working with the search results, you can free the chain itself, rather than individual structures within the chain.

For more information, see Chapter 5, “Searching a LDAP Directory.”

See Also

`ldap_result()`, `ldap_search_ext()`, `ldap_search_ext_s()`, `ldap_next_entry()`

ldap_first_message()

The `ldap_first_message()` function returns a pointer to the first `LDAPMessage` structure in a chain of search results.

Syntax

```
#include <ldap.h>
LDAPMessage * ldap_first_message( LDAP *ld, LDAPMessage *res );
```

Parameters

This function has the parameters listed in Table 17-87.

Table 17-87 `ldap_first_message()` Function Parameters

Parameter	Description
<code>ld</code>	Pointer to an LDAP structure containing information about the connection to the LDAP server.
<code>res</code>	Chain of search results, represented by a pointer to the <code>LDAPMessage</code> structure.

Returns

One of the following values:

- If successful, returns the pointer to the first LDAPMessage structure in the chain of search results.
- If no LDAPMessage structures are in the chain or if the function is unsuccessful, returns a NULLMSG.

Description

The `ldap_first_message()` function returns a pointer to the first LDAPMessage structure in a chain of search results. You can use this function in conjunction with the `ldap_next_message()` function to iterate through the chain of search results. You can also call the `ldap_msgtype()` function to determine if each message contains a matching entry (a message of the type `LDAP_RES_SEARCH_ENTRY`) or a search reference (a message of the type `LDAP_RES_SEARCH_REFERENCE`).

CAUTION Do not free the LDAPMessage structure returned by this function. Because this is a structure within a chain of search results, freeing this structure will free part of the chain of search results. When you are done working with the search results, you can free the chain itself, rather than individual structures within the chain.

For more information, see Chapter 5, “Searching a LDAP Directory.”

Example

See Code Example 5-4 and Code Example 5-7 in Chapter 5, “Searching a LDAP Directory.”

See Also

`ldap_result()`, `ldap_search_ext()`, `ldap_search_ext_s()`,
`ldap_next_message()`, `ldap_first_entry()`, `ldap_first_reference()`

ldap_first_reference()

The `ldap_first_reference()` function returns a pointer to the LDAPMessage structure representing the first search reference in a chain of search results.

Syntax

```
#include <ldap.h>
LDAPMessage * ldap_first_reference(LDAP *ld, LDAPMessage *res );
```

Parameters

This function has the parameters listed in Table 17-88.

Table 17-88 ldap_first_reference() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
res	Chain of search results, which are represented by a pointer to the LDAPMessage structure.

Returns

One of the following values:

- If successful, returns the pointer to the first LDAPMessage structure of the type LDAP_RES_SEARCH_REFERENCE in the chain of search results.
- If no LDAPMessage structures of the type LDAP_RES_SEARCH_REFERENCE are in the chain of the search results or if the function is unsuccessful, returns a NULLMSG.

Description

The ldap_first_reference() function returns a pointer to the LDAPMessage structure representing the first search reference in a chain of search results. Search references are in messages of the type LDAP_RES_SEARCH_REFERENCE. You can use this function in conjunction with the ldap_next_reference() function to iterate through the search references in a chain of search results. These functions skip over any messages in the chain that do not have the type LDAP_RES_SEARCH_REFERENCE.

CAUTION Do not free the LDAPMessage structure returned by this function. Because this is a structure within a chain of search results, freeing this structure will free part of the chain of search results. When you are done working with the search results, you can free the chain itself, rather than individual structures within the chain.

For more information, see Chapter 5, “Searching a LDAP Directory.”

See Also

ldap_result(), ldap_search_ext(), ldap_search_ext_s(),
ldap_next_reference(), ldap_first_message()

ldap_first_searchobj()

The ldap_first_searchobj() function returns the first search preference configuration from a data structure defined in the list solist.

Syntax

```
#include <ldap.h>
ldap_searchobj * ldap_first_searchobj( struct ldap_searchobj *solist );
```

Parameters

This function has the parameter listed in Table 17-89.

Table 17-89 ldap_first_searchobj() Function Parameter

Parameter	Description
solist	Pointer to the search preference data structures, typically obtained by calling ldap_init_searchprefs().

Description

ldap_first_searchobj provides access to LDAP search preference configuration data. LDAP search preference configurations are typically used by LDAP client programs to specify which attributes a user may search by, the labels for the attributes, and the LDAP filters and scopes associated with those searches.

See Also

ldap_init_searchprefs(), ldap_init_searchprefs_buf()

ldap_first_tmplcol()

The ldap_first_tmplcol() function returns a pointer to the first item within a template.

Syntax

```
#include <disptmpl.h>
struct ldap_tmplitem * ldap_first_tmplcol( struct ldap_disptmpl *tmpl,
      struct ldap_tmplitem *row );
```

Parameters

This function has the parameters listed in Table 17-90.

Table 17-90 ldap_first_tmplcol() Function Parameters

Parameter	Description
tmpl	The name of the template to be retrieved.
row	The row in which the item is to be retrieved from.

Returns

- If successful, a pointer to an `ldap_tmplitem` structure.
- A NULL pointer on error.

Description

`ldap_first_tmplcol()` returns a pointer to the first item (in the first column) of the row, defined by `row`, within the template defined by `tmpl`.

See Also

`ldap_first_tmplrow()`, `ldap_next_tmplcol()`

ldap_first_tmplrow()

The `ldap_first_tmplrow()` function returns a pointer to the first row of items in a template.

Syntax

```
#include <disptmpl.h>
struct ldap_tmplitem * ldap_first_tmplrow( struct ldap_disptmpl *tmpl );
```

Parameters

This function has the parameter listed in Table 17-91.

Table 17-91 ldap_first_tmplrow() Function Parameter

Parameter	Description
tmpl	The name of the template to be retrieved.

Returns

- If successful, a pointer to an `ldap_tmplitem` structure.
- A NULL pointer on error.

ldap_free_friendlymap()

See Also

ldap_next_tmplrow(), ldap_first_tmplcol()

ldap_free_friendlymap()

The ldap_free_friendlymap() function frees the FriendlyMap structures allocated by the ldap_friendly_name() function when no more calls to it are to be made.

Syntax

```
#include <ldap.h>
void ldap_free_friendlymap( FriendlyMap *map );
```

Parameters

This function has the parameter listed in Table 17-92.

Table 17-92 ldap_free_friendlymap() Function Parameter

Parameter	Description
map	Pointer to the FriendlyMap mapping structure in memory.

Example

Code Example 17-15 frees memory allocated by the ldap_friendly_name() function.

Code Example 17-15 ldap_free_friendlymap() Code Example

```
#include <ldap.h>
#include <stdio.h>
...
FriendlyMap map = NULL;
char *map_file = "/u/mozilla/ldapsdk/examples/ldapfriendly";
char *unfriendly_name = "IS";
char *friendly_name;
...
/* Read the ldapfriendly file into the map in memory */
friendly_name = ldap_friendly_name( map_file, unfriendly_name, &map );
printf( "Friendly Name for %s: %s\n", unfriendly_name, friendly_name );

/* Since file is in memory, no need to reference it in subsequent calls */
friendly_name = ldap_friendly_name( NULL, "VI", &map );
printf( "Friendly Name for VI: %s\n", friendly_name );
...
ldap_free_friendlymap( map );
...
```


See Also
ldap_friendly_name(), FriendlyMap

ldap_free_searchprefs()

The ldap_free_searchprefs() function disposes of the data structures allocated by ldap_init_searchprefs().

Syntax
#include <ldap.h>
void ldap_free_searchprefs(struct ldap_searchobj *solist);

Parameters
This function has the parameter listed in Table 17-93.

Table 17-93 ldap_free_searchprefs() Function Parameter

Parameter	Description
solist	Pointer to the data structures, typically obtained by calling ldap_init_searchprefs().

See Also
ldap_init_searchprefs(), ldap_init_searchprefs_buf()

ldap_free_sort_keylist()

The ldap_free_sort_keylist() function frees the structures allocated by the ldap_create_sort_keylist() function.

Syntax
#include <ldap.h>
void ldap_free_sort_keylist (LDAPsortkey **sortKeyList);

Parameters
This function has the parameter listed in Table 17-94.

Table 17-94 ldap_free_sort_keylist() Function Parameter

Parameter	Description
sortKeyList	Array of LDAPsortkey structures that you want to free from memory.

Description

The ldap_free_sort_keylist() function frees the array of LDAPsortkey structures allocated by the ldap_create_sort_keylist() function. When done sorting results, call this function to free the memory that you have allocated. This function must be called after the ldap_create_sort_control() function has completed.

See Also

ldap_create_sort_keylist(), ldap_create_sort_control()

ldap_free_templates()

The ldap_free_templates() function disposes of the templates allocated by ldap_init_templates().

Syntax

```
#include <disptmpl.h>
void ldap_free_templates( struct ldap_disptmpl *tmplist );
```

Parameters

This function has the parameter listed in Table 17-95.

Table 17-95 ldap_free_templates() Function Parameter

Parameter	Description
tmplistp	Pointer to a list of template data structures.

Description

ldap_free_templates() releases the templates allocated by ldap_init_templates(). Each template defined in the file is an ldap_disptmpl structure.

ldap_free_urldesc()

The `ldap_free_urldesc()` function frees memory allocated by the `ldap_url_parse()` function.

Syntax

```
#include <ldap.h>
void ldap_free_urldesc( LDAPURLDesc *ludp );
```

Parameters

This function has the parameter listed in Table 17-96.

Table 17-96 ldap_free_urldesc() Function Parameter

Parameter	Description
<code>ludp</code>	Pointer to a <code>LDAPURLDesc</code> structure.

Description

For more information, see “Freeing the Components of an LDAP URL” on page 206 of Chapter 9, “Working with LDAP URLs.”

Example

Code Example 17-16 parses an LDAP URL and then frees the `LDAPURLDesc` structure from memory after verifying that the LDAP URL is valid.

Code Example 17-16 ldap_free_urldesc() Code Example

```
#include <stdio.h>
#include <ldap.h>
...
char *my_url =
"ldap://ldap.example.com:5000/dc=example,dc=com?cn,mail,telephoneNumber?sub
? \
(sn=Jensen)";
LDAPURLDesc *ludpp;
int res, i;
...
if ( ( res = ldap_url_parse( my_url, &ludpp ) ) != 0 ) {
    switch( res ){
        case LDAP_URL_ERR_NOTLDAP:
            printf( "URL does not begin with \"ldap://\"\\n" );
            break;
        case LDAP_URL_ERR_NODN:
            printf( "URL does not contain a distinguished name\\n" );
            break;
        case LDAP_URL_ERR_BADSCOPE:
```

ldap_friendly_name()

Code Example 17-16 ldap_free_urldesc() Code Example (*Continued*)

```
        printf( "URL contains an invalid scope\n" );
        break;
    case LDAP_URL_ERR_MEM:
        printf( "Not enough memory\n" );
        break;
    default:
        printf( "Unknown error\n" );
    }
    return( 1 );
}
printf( "URL is a valid LDAP URL\n" );
ldap_free_urldesc( ludpp );
...
```

See Also
ldap_url_parse()

ldap_friendly_name()

The ldap_friendly_name() function maps a set of standard identifiers to their user-friendly counterparts. For example, you can represent the list of two-letter state codes (CA, IA) with their corresponding state names (California, Iowa), or map country ISO codes to the full country names.

NOTE ldapfriendly, located in *DSRK_base/lib/ldapsdk/etc*, is a sample file that maps two letter country codes to their full names. It can be used in context with ldap_friendly_name().

Syntax
#include <ldap.h>
char * ldap_friendly_name(char *filename, char *uname,
 FriendlyMap *map);

Parameters
This function has the parameters listed in Table 17-97.

Table 17-97 ldap_friendly_name() function parameters

Parameter	Description
filename	Name of the file mapping the standard identifiers to the user-friendly names.

Table 17-97 ldap_friendly_name() function parameters (*Continued*)

Parameter	Description
uname	Standard identifier name for which you want to find the user-friendly name.
map	Pointer to the FriendlyMap mapping in memory. Initialize this pointer to NULL on the first call, then use it during subsequent calls so that the mapping file does not need to be read again.

Returns

One of the following values:

- If successful, returns the user-friendly name for the specified identifier.
- If unsuccessful (for example, if the file cannot be read, if the file is in a bad format, or if the map parameter is set to NULL), returns the original identifier (the value passed as the uname parameter).

Description

This function relies on the existence of a text file mapping standard identifiers to user-friendly names. The names in the file are tab-delimited, as shown in Code Example 17-17.

Code Example 17-17 Unfriendly to Friendly Name Mapping File

```
<unfriendly_name>      <friendly_name>
AD      Andorra
AE      United Arab Emirates
AF      Afghanistan
AG      Antigua and Barbuda
AI      Anguilla
```

Example

Code Example 17-18 reads in a map of user-friendly names and prints the name corresponding to the standard identifier *IS*.

Code Example 17-18 ldap_friendly_name() Code Example

```
#include <ldap.h>
#include <stdio.h>
...
FriendlyMap map = NULL;
char *map_file = "/u/mozilla/ldapsdk/examples/ldapfriendly";
```

Code Example 17-18 ldap_friendly_name() Code Example (*Continued*)

```

char *unfriendly_name = "IS";
char *friendly_name;
...
/* Read the ldapfriendly file into the map in memory */
friendly_name = ldap_friendly_name( map_file, unfriendly_name, &map );
printf( "Friendly Name for %s: %s\n", unfriendly_name, friendly_name );

/* Since file is in memory, no need to reference it in subsequent calls */
friendly_name = ldap_friendly_name( NULL, "VI", &map );
printf( "Friendly Name for VI: %s\n", friendly_name );
...

```

See Also

ldap_free_friendlymap(), FriendlyMap

ldap_get_dn()

The ldap_get_dn() routine returns the DN for an entry in a chain of search results.

Syntax

```

#include <ldap.h>
char * ldap_get_dn( LDAP *ld, LDAPMessage *entry );

```

Parameters

This function has the parameters listed in Table 17-98.

Table 17-98 ldap_get_dn() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
entry	Pointer to an entry in a chain of search results, as returned by the ldap_first_entry() and ldap_next_entry() functions.

Returns

One of the following values:

- If successful, returns the DN for the specified entry.

- If unsuccessful, returns a NULL and sets the appropriate error code in the LDAP structure. To get the error code, call the `ldap_get_lderrno()` function.

See Chapter 18, “Result Codes” for a complete listing of error codes.

Description

You can get an entry from a chain of search results by calling the `ldap_first_entry()` and `ldap_next_entry()` functions.

For more information, see “Getting the Distinguished Name of an Entry” on page 125 of Chapter 5, “Searching a LDAP Directory.”

Example

Code Example 17-19 prints the DN for each entry found in a search.

Code Example 17-19 ldap_get_dn() Code Example

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result, *e;
char *dn;
char *my_searchbase = "dc=example,dc=com";
char *my_filter = "(sn=Jensen)";
...
/* Search the directory */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, my_filter,
    NULL, 0, &result ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_search_s" );
    return( 1 );
}

/* For each matching entry found, print out the name of the entry.*/
for ( e = ldap_first_entry( ld, result ); e != NULL;
    e = ldap_next_entry( ld, e ) ) {
    if ( ( dn = ldap_get_dn( ld, e ) ) != NULL ) {
        printf( "dn: %s\n", dn );
        /* Free the memory used for the DN when done */
        ldap_memfree( dn );
    }
}
/* Free the result from memory when done. */
ldap_msgfree( result );
...
```

See Also

`ldap_first_entry()`, `ldap_next_entry()`, `ldap_get_lderrno()`

ldap_get_entry_controls()

The `ldap_get_entry_controls()` function gets the LDAP controls included with a directory entry in a set of search results.

Syntax

```
#include <ldap.h>
int ldap_get_entry_controls( LDAP *ld, LDAPMessage *entry,
    LDAPControl ***serverctrlsp );
```

Parameters

This function has the parameters listed in Table 17-99.

Table 17-99 ldap_get_entry_controls() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
entry	Pointer to an LDAPMessage structure representing an entry in a chain of search results.
serverctrlsp	Pointer to an array of LDAPControl structures, which represent the LDAPv3 server controls returned by the server.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_DECODING_ERROR if an error occurred when decoding the BER-encoded results from the server.
- LDAP_PARAM_ERROR if an invalid parameter was passed to the function.
- LDAP_NO_MEMORY if memory cannot be allocated.

Description

The `ldap_get_entry_controls()` function retrieves the LDAPv3 controls included in a directory entry in a chain of search results. The LDAP controls are specified in an array of LDAPControl structures with each structure representing one LDAP control. Other controls are returned with results sent from the server. You can call `ldap_parse_result()` to retrieve those controls.

ldap_getfilter_free()

The ldap_getfilter_free() function frees the memory used by a filter set.

Syntax

```
#include <ldap.h>
void ldap_getfilter_free( LDAPFiltDesc *lfdp );
```

Parameters

This function has the parameter listed in Table 17-100.

Table 17-100ldap_getfilter_free() Function Parameter

Parameter	Description
lfdp	Pointer to a LDAPFiltDesc structure.

Description

Once you call ldap_getfilter_free(), the LDAPFiltDesc structure is no longer valid and cannot be used again.

For more information, see “Freeing Filters from Memory” on page 145 of Chapter 17, “Function Reference.”

Example

Code Example 17-20 frees the LDAPFiltDesc structure from memory after all searches are completed.

Code Example 17-20 ldap_getfilter_free Code Example

```
#include <ldap.h>
...
LDAPFiltDesc *lfdp;
char *filter_file = "myfilters.conf";
...
/* Read the filter configuration file into an LDAPFiltDesc structure */
lfdp = ldap_init_getfilter( filter_file );
...
/* Retrieve filters and perform searches */
...
/* Free the configuration file (the LDAPFiltDesc structure) */
ldap_getfilter_free( lfdp );
...
```

See Also

ldap_init_getfilter(), ldap_init_getfilter_buf()

ldap_getfirstfilter()

The ldap_getfirstfilter() function retrieves the first filter that is appropriate for a given value.

Syntax

```
#include <ldap.h>
LDAPFiltInfo * ldap_getfirstfilter( LDAPFiltDesc *lfdp,
    char *tagpat, char *value );
```

Parameters

This function has the parameters listed in Table 17-101.

Table 17-101 ldap_getfirstfilter() Function Parameters

Parameter	Description
lfdp	Pointer to an LDAPFiltDesc structure.
tagpat	Regular expression for a tag in the filter configuration.
value	Value for which to find the first appropriate filter.

Returns

One of the following values:

- If successful, returns a pointer to an LDAPFiltInfo structure.
- If no more filters can be returned, returns a NULL.
- If unsuccessful, returns a NULL.

Description

For more information, see “Retrieving Filters” on page 143 of Chapter 6, “Using Filter Configuration Files.”

Example

Code Example 17-21 is based on the `getfilt` command-line program example provided with the LDAP SDK for C. The program prompts the user to enter search criteria and, based on the criteria entered, it retrieves filters that match the criteria. Code Example 17-21 uses the filter configuration file shown in “The Filter Configuration File Syntax” on page 140 of Chapter 6, “Using Filter Configuration Files.”

Code Example 17-21 `ldap_getfirstfilter()` Code Example

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

...
LDAP *ld;
LDAPMessage *result, *e;
BerElement *ber;
char *a, *dn;
char **vals;
int i;
LDAPFiltDesc *ldfp;
LDAPFiltInfo *ldfi;
char buf[ 80 ]; /* Contains the search criteria */
int found;

...
/* Load the filter configuration file into an LDAPFiltDesc structure */
if ( ( ldfp = ldap_init_getfilter( "myfilters.conf" ) ) == NULL ) {
    perror( "Cannot open filter configuration file" );
    return( 1 );
}

/* Read a string to search for */
printf( "Enter a string to search for: " );
gets( buf );
if ( strlen( buf ) == 0 ) {
    fprintf( stderr, "usage: %s search-string\n", argv[ 0 ] );
    return( 1 );
}

/* Select a filter to use when searching for the value in buf */
found = 0;
for ( ldfi = ldap_getfirstfilter( ldfp, "people", buf ); ldfi != NULL; ldfi
= ldap_getnextfilter( ldfp ) ) {

    /* Use the selected filter to search the directory */
    if ( ldap_search_s( ld, "dc=example,dc=com, ldfi->lfi_scope,
ldfi->lfi_filter, NULL, 0, &result ) != LDAP_SUCCESS ) {
        ldap_perror( ld, "ldap_search_s" );
        return( 1 );
    } else {

        /* Once a filter gets results back, stop iterating through the different
filters */
        if ( ( found = ldap_count_entries( ld, result ) > 0 ) ) {
```

Code Example 17-21 ldap_getfirstfilter() Code Example (*Continued*)

```

        break;
    } else {
        ldap_msgfree( result );
    }
}

if ( found == 0 ) {
    printf( "No matching entries found.\n" );
} else {
    printf( "Found %d %s match%s for \"%s\"\n\n", found,
        ldfi->lfi_desc, found == 1 ? "" : "es", buf );
}

ldap_msgfree( result );
ldap_getfilter_free( ldfp );
...

```

See Also

ldap_init_getfilter(), ldap_init_getfilter_buf(), ldap_getnextfilter()

ldap_get_lang_values()

CAUTION This function is deprecated and should not be used. It is included in ldap-deprecated.h for backward-compatibility. Please use ldap_get_values().

The ldap_get_lang_values() function returns a NULL-terminated array of an attribute's string values that match a specified language subtype.

Syntax

```

#include <ldap.h>
char ** ldap_get_lang_values( LDAP *ld, LDAPMessage *entry,
    const char *target, char **type );

```

Parameters

This function has the parameters listed in Table 17-102.

Table 17-102ldap_get_lang_values() Function Parameters

Parameters	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
entry	Entry retrieved from the directory.
target	Attribute type (including an optional language subtype) that you want to retrieve the values of.
type	Pointer to a buffer that returns the attribute type retrieved by this function.

Returns

One of the following values:

- If successful, returns a NULL-terminated array of the attribute’s values.
- If unsuccessful or if no such attribute exists in the entry, returns a NULL and sets the appropriate error code in the LDAP structure. To get the error code, call the ldap_get_lderrno() function.

See Chapter 18, “Result Codes” for a complete listing of error codes.

Description

Unlike the ldap_get_values() function, if a language subtype is specified, this function first attempts to find and return values that match that subtype (for example, cn; lang-en). If you want to retrieve binary data from an attribute, call the ldap_get_lang_values_len() function.

See Also

ldap_first_entry(), ldap_first_attribute(), ldap_get_lang_values_len(), ldap_get_values(), ldap_next_entry(), ldap_next_attribute()

ldap_get_lang_values_len()

CAUTION This function is deprecated and should not be used. It is included in ldap-deprecated.h for backward-compatibility. Please use ldap_get_values_len().

The `ldap_get_lang_values_len()` function returns a NULL-terminated array of pointers to `berval` structures, each containing the length and pointer to a binary value of an attribute for a given entry.

Syntax

```
#include <ldap-deprecated.h>
struct berval ** ldap_get_lang_values_len( LDAP *ld,
    LDAPMessage *entry, const char *target, char **type );
```

Parameters

This function has the parameters listed in Table 17-103.

Table 17-103 `ldap_get_lang_values_len()` Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
entry	Result returned by the <code>ldap_result()</code> or <code>ldap_search_s()</code> function.
target	Attribute returned by <code>ldap_first_attribute()</code> or <code>ldap_next_attribute()</code> , or the attribute as a literal string, such as "jpegPhoto" or "audio".
type	Pointer to a buffer that returns the attribute type retrieved by this function.

Returns

One of the following values:

- If successful, returns a NULL-terminated array of pointers to `berval` structures, which, in turn, contain pointers to the attribute's binary values.
- If unsuccessful or if no such attribute exists in the entry, returns a NULL and sets the appropriate error code in the LDAP structure. To get the error code, call the `ldap_get_lderrno()` function.

See Chapter 18, "Result Codes" for a complete listing of error codes.

Description

Use the `ldap_get_lang_values()` routine if the attribute values are string values.

For more information, see "Getting the Values of an Attribute" on page 128 of Chapter 5, "Searching a LDAP Directory."

See Also

ldap_first_attribute(), ldap_first_entry(), ldap_get_lang_values(),
 ldap_next_entry(), ldap_next_attribute()

ldap_get_lderrno()

The ldap_get_lderrno() function gets information about the last error that occurred for an LDAP operation.

Syntax

```
#include <ldap.h>
int ldap_get_lderrno( LDAP *ld, char **m, char **s );
```

Parameters

This function has the parameters listed in Table 17-104.

Table 17-104 ldap_get_lderrno() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
m	In the event of an LDAP_NO_SUCH_OBJECT error return, this parameter contains the portion of the DN that identifies an existing entry. (See “Receiving the Matching Portion of a DN” on page 92 of Chapter 4, “Using the LDAP API.”)
s	The text of the error message.

Returns

The LDAP error code for the last operation. See Chapter 18, “Result Codes” for a complete listing of error codes.

Description

ldap_get_lderrno() gets information about the last error that occurred for an LDAP operation. You can also call this function to get error codes for functions that do not return errors (such as ldap_next_attribute()).

For more information, see “Getting Information About an Error” on page 88 of Chapter 4, “Using the LDAP API.”

Example

Code Example 17-22 attempts to add a new user to the directory. If the entry identified by a DN does not exist, the server returns the portion of the DN that matches an existing entry; this is the variable `matched`. (See “Receiving the Matching Portion of a DN” on page 92 of Chapter 4, “Using the LDAP API” for details.)

Code Example 17-22 ldap_get_lderrno() Code Example

```
#include <ldap.h>
LDAP *ld;
char *dn = "uid=bjensen,ou=People,dc=example,dc=com";
LDAPMod **list_of_attrs;
char *matched;
int rc;
...
if ( ldap_add_s( ld, dn, list_of_attrs ) != LDAP_SUCCESS ) {
    rc = ldap_get_lderrno( ld, &matched, NULL );
    return( rc );
}
...
```

In Code Example 17-22, if no organizational unit named `People` exists, the `matched` variable is set to `dc=example,dc=com`.

See Also

ldap_err2string(), ldap_perror(), ldap_result2error(), ldap_set_lderrno()

ldap_getnextfilter()

The `ldap_getnextfilter()` function retrieves the next filter that is appropriate for a given value.

Syntax

```
#include <ldap.h>
LDAPFiltInfo * ldap_getnextfilter( LDAPFiltDesc *lfdp );
```

Parameters

This function has the parameter listed in Table 17-105.

Table 17-105 ldap_getnextfilter() Function Parameter

Parameter	Description
lfdp	Pointer to an LDAPFiltDesc structure.

Returns

One of the following values:

- If successful, returns a pointer to an LDAPFiltInfo structure.
- If no more filters can be returned, returns a NULL.
- If unsuccessful, returns a NULL.

Description

Call this function to get subsequent filters after calling ldap_getfirstfilter().

For more information, see “Retrieving Filters” on page 143 of Chapter 6, “Using Filter Configuration Files.”

Example

See Code Example 17-21 on page 451 under ldap_getfirstfilter().

See Also

ldap_getfirstfilter()

ldap_get_option()

The function ldap_get_option() retrieves session preferences from an LDAP structure.

Syntax

```
#include <ldap.h>
int ldap_get_option( LDAP *ld, int option, void *optdata );
```

Parameters

This function has the parameters listed in Table 17-106.

Table 17-106ldap_get_option() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
option	Option that you want to retrieve. This parameter must be set to one of the option values, as detailed in Table 17-107 on page 458.
optdata	Pointer to the buffer in which the value of the option will be put.

Table 17-107 describes the options that you can retrieve with ldap_get_option().

Table 17-107Options for the ldap_get_options() Function

Option	Description
LDAP_OPT_API_FEATURE_INFO	Retrieves information about the revision of a supported LDAP feature. This option is READ-ONLY and cannot be set. The data type for the optdata parameter is (LDAPAPIFeatureInfo *).
LDAP_OPT_API_INFO	Retrieves information about the API implementation at execution time (API version, protocol version, the names of supported API extensions with their vendor name version, etc.). For details on the structure returned, refer to the ldap.h header file. This option is READ-ONLY and cannot be set. The data type for the optdata parameter is (LDAPAPIInfo *).
LDAP_OPT_CLIENT_CONTROLS	Pointer to an array of LDAPControl structures representing the LDAPv3 client controls you want sent with every request by default. The data type for the optdata parameter is (LDAPControl ***).
LDAP_OPT_DESC	Socket descriptor underlying the main LDAP connection. The LBER_SOCKET data type depends on the platform that you are using: • int in UNIX. • SOCKET in Windows. The data type for the optdata parameter is (LBER_SOCKET *).

Table 17-107Options for the ldap_get_options() Function (*Continued*)

Option	Description
LDAP_OPT_DEREF	Determines how aliases work during a search. optdata can be one of the following values: - LDAP_DEREF_NEVER specifies that aliases are never dereferenced. - LDAP_DEREF_SEARCHING specifies that aliases are dereferenced when searching under the base object (but not when finding the base object). - LDAP_DEREF_FINDING specifies that aliases are dereferenced when finding the base object (but not when searching under the base object). - LDAP_DEREF_ALWAYS specifies that aliases are always dereferenced when finding and searching under the base object. The data type for the optdata parameter is (int *).
LDAP_OPT_DNS_FN_PTRS	DEPRECATED OPTION: Lets you use alternate DNS functions for getting the host entry of the LDAP server. The data type for the optdata parameter is (struct ldap_dns_fns *).
LDAP_OPT_ERROR_NUMBER	Retrieves the result code for the most recent LDAP error that occurred in this session. The data type for the optdata parameter is (int *).
LDAP_OPT_ERROR_STRING	Retrieves the error message returned with the result code for the most recent LDAP error that occurred in this session. The data type for the optdata parameter is (char **).
LDAP_OPT_EXTRA_THREAD_FN_PTRS	Lets you specify the locking and semaphore functions that you want called when getting results from the server. (See Chapter 14, “Writing Multithreaded Clients” for details.) The data type for the optdata parameter is (struct ldap_extra_thread_fns *).
LDAP_OPT_HOST_NAME	Sets the host name (or list of hosts) for the primary LDAP server. The data type for the optdata parameter is (char **).
LDAP_OPT_IO_FN_PTRS	DEPRECATED OPTION: Lets you use alternate communication stacks. The data type for the optdata parameter is (struct ldap_io_fns *).
LDAP_OPT_MATCHED_DN	Gets the matched DN value returned with the most recent LDAP error that occurred for this session. The data type for the optdata parameter is (char **)

Table 17-107Options for the ldap_get_options() Function (*Continued*)

Option	Description
LDAP_OPT_MEMALLOC_FN_PTRS	Gets a pointer to the callback structure which you previously set. The data type for the optdata parameter is (struct ldap_memalloc_fnslldap_io_fnsl *).
LDAP_OPT_PROTOCOL_VERSION	Version of the protocol supported by your client. You can specify either LDAP_VERSION2 or LDAP_VERSION3. If no version is set, the default is LDAP_VERSION2. In order to use LDAPv3 features, you need to set the protocol version to LDAP_VERSION3. The data type for the optdata parameter is (int *).
LDAP_OPT_REBIND_ARG	Lets you set the last argument passed to the routine specified by LDAP_OPT_REBIND_FN. You can also set this option by calling the ldap_set_rebind_proc() function. The data type for the optdata parameter is (void *).
LDAP_OPT_REBIND_FN	Lets you set the routine to be called when you need to authenticate a connection with another LDAP server (for example, during the course of following a referral). You can also set this option by calling the ldap_set_rebind_proc() function. The data type for the optdata parameter is (LDAP_REBINDPROC_CALLBACK *).
LDAP_OPT_RECONNECT	If the connection to the server is lost, determines whether or not the same connection handle should be used to reconnect to the server. By default, this option is off. For details, see “Handling Failover” on page 100 of Chapter 4, “Using the LDAP API.” optdata can be one of the following values: • LDAP_OPT_ON specifies that the same connection handle can be used to reconnect to the server. • LDAP_OPT_OFF specifies that you want to create a new connection handle to connect to the server. The data type for the optdata parameter is (int *).
LDAP_OPT_REFERRALS	Determines whether or not the client should follow referrals. By default, the client follows referrals. optdata can be one of the following values: • LDAP_OPT_ON specifies that the server should follow referrals. • LDAP_OPT_OFF specifies that the server should not follow referrals. The data type for the optdata parameter is (int *).

Table 17-107Options for the ldap_get_options() Function (*Continued*)

Option	Description
LDAP_OPT_REFERRAL_HOP_LIMIT	Maximum number of referrals the client should follow in a sequence. In other words, the client can only be referred this number of times before it gives up. By default, the maximum number of referrals that the client can follow in a sequence is 5 for the initial connection. This limit does not apply to individual requests that generate multiple referrals in parallel. The data type for the optdata parameter is (int *).
LDAP_OPT_RESTART	Determines whether or not LDAP I/O operations should be restarted automatically if they are prematurely aborted. optdata can be one of the following values: - LDAP_OPT_ON specifies that I/O operations should be restarted automatically. - LDAP_OPT_OFF specifies that I/O operations should not be restarted automatically. The data type for the optdata parameter is (int *).
LDAP_OPT_SERVER_CONTROLS	Pointer to an array of LDAPControl structures representing the LDAPv3 server controls you want sent with every request by default. Typically, since controls are specific to the type of request, you may want to pass the controls using operation-specific functions (such as ldap_add_ext()) instead. The data type for the optdata parameter is (LDAPControl ***).
LDAP_OPT_SIZELIMIT	Maximum number of entries that should be returned by the server in search results. The LDAP server may impose a smaller size limit than the limit you specify as the server administrator also has the ability to set this limit. The data type for the optdata parameter is (int *).
LDAP_OPT_SSL	Determines whether or not SSL is enabled. optdata can be one of the following values: - LDAP_OPT_ON specifies that SSL is enabled. - LDAP_OPT_OFF specifies that SSL is disabled. The data type for the optdata parameter is (int *).
LDAP_OPT_THREAD_FN_PTRS	Lets you specify the thread function pointers. (See Chapter 14, “Writing Multithreaded Clients” for details.) The data type for the optdata parameter is (struct ldap_thread_fns *).

Table 17-107Options for the ldap_get_options() Function (*Continued*)

Option	Description
LDAP_OPT_TIMELIMIT	Maximum number of seconds that should be spent by the server when answering a search request. The LDAP server may impose a shorter time limit than the limit you specify as the server administrator also has the ability to set this limit. The data type for the optdata parameter is (int *).
LDAP_X_OPT_EXTIO_FN_PTRS	Extended I/O function callback option.
LDAP_X_OPT_CONNECT_TIMEOUT	Value of a timeout (expressed in milliseconds) for nonblocking connect call. The data type for the optdata parameter is (int *).
LDAP_X_OPT_SOCKETBUF	Socket buffer structure associated to the LDAP connection.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- -1 if unsuccessful.

Description

For more information, see “Retrieving SDK Information” on page 85 of Chapter 4, “Using the LDAP API.”

Examples

Code Example 17-23 gets the session preference for the maximum number of entries to be returned from search operations.

Code Example 17-23 ldap_get_option() Code Example

```
#include <ldap.h>
...
LDAP *ld;
int max_ret;
...
/* Get the maximum number of entries returned */
if (ldap_get_option( LDAP_OPT_SIZELIMIT, &max_ret) != LDAP_SUCCESS) {
    ldap_perror( ld, "ldap_get_option" );
    return( 1 );
}
```

Code Example 17-24 contains two small sections of code that show how to use the LDAP_OPT_API_FEATURE_INFO and the LDAP_OPT_API_INFO options, respectively.

Code Example 17-24 LDAP_OPT_API_FEATURE_INFO and LDAP_OPT_API_INFO Usage

```
LDAPFeatureInfo ldfi;
ldfi.ldapaif_info_version = LDAP_FEATURE_INFO_VERSION;
ldfi.ldapaif_name = "VIRTUAL_LIST_VIEW";
if (ldap_get_option(NULL, LDAP_OPT_API_FEATURE_INFO, &ldfi)==0) {
/* use the info here */ }

LDAPInfo ldai;
ldai.ldapiai_info_version = LDAP_API_INFO_VERSION;
if (ldap_get_option( NULL, LDAP_OPT_API_INFO, &ldia ) == 0) {
/* use the ldai info here */
}
```

See Also
ldap_init(), ldap_set_option()

ldap_get_values()

The ldap_get_values() function returns a NULL-terminated array of an attribute’s string values for a given entry.

Syntax
#include <ldap.h>
char ** ldap_get_values(LDAP *ld, LDAPMessage *entry,
const char *target);

Parameters
This function has the parameters listed in Table 17-108.

Table 17-108ldap_get_values() Function Parameters

Parameters	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
entry	Result returned by the ldap_result() or ldap_search_s() function.
target	Attribute returned by ldap_first_attribute() or ldap_next_attribute(), or the attribute as a literal string, such as jpegPhoto or audio.

Returns

One of the following values:

- If successful, returns a NULL-terminated array of the attribute's values.
- If unsuccessful or if no such attribute exists in the entry, returns a NULL and sets the appropriate error code in the LDAP structure. To get the error code, call the `ldap_get_lderrno()` function.

See Chapter 18, “Result Codes” for a complete listing of error codes.

Description

Use the `ldap_get_values_len()` function if the attribute values are binary.

For more information, see “Getting the Values of an Attribute” on page 128 of Chapter 5, “Searching a LDAP Directory.”

Example

Code Example 17-25 gets and prints the values of an attribute in an entry. This example assumes that all attributes have string values.

Code Example 17-25 `ldap_get_values()` Code Example

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result, *e;
BerElement *ber;
char *a;
char **vals;
char *my_searchbase = "dc=example,dc=com";
char *my_filter = "(sn=Jensen)";
int i;
...
/* Search the directory */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, my_filter,
    NULL, 0, &result ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_search_s" );
    return( 1 );
}

/* Get the first matching entry.*/
e = ldap_first_entry( ld, result );

/* Get the first matching attribute */
a = ldap_first_attribute( ld, e, &ber );

/* Get the values of the attribute */
if ( ( vals = ldap_get_values( ld, e, a ) ) != NULL ) {
    for ( i = 0; vals[i] != NULL; i++ ) {
        /* Print the name of the attribute and each value */
    }
}
```


Code Example 17-25 ldap_get_values() Code Example (*Continued*)

```

    printf( "%s: %s\n", a, vals[i] );
}
/* Free the attribute values from memory when done. */
ldap_value_free( vals );
}
...
```

See Also

ldap_first_entry(), ldap_first_attribute(), ldap_get_values_len(),
 ldap_next_entry(), ldap_next_attribute()

ldap_get_values_len()

The `ldap_get_values_len()` function returns a NULL-terminated array of pointers to `berval` structures, each containing the length and pointer to a binary value of an attribute for a given entry.

Syntax

```
#include <ldap.h>
struct berval ** ldap_get_values_len( LDAP *ld,
    LDAPMessage *entry, const char *target );
```

Parameters

This function has the parameters listed in Table 17-109.

Table 17-109 ldap_get_values_len() Function Parameters

Parameter	Description
<code>ld</code>	Pointer to an LDAP structure containing information about the connection to the LDAP server.
<code>entry</code>	Result returned by the <code>ldap_result()</code> or <code>ldap_search_s()</code> function.
<code>target</code>	Attribute returned by <code>ldap_first_attribute()</code> or <code>ldap_next_attribute()</code> , or the attribute as a literal string, such as <code>jpegPhoto</code> or <code>audio</code> .

Returns

One of the following values:

- If successful, returns a NULL-terminated array of pointers to berval structures, which in turn contains pointers to the attribute's binary values.
- If unsuccessful or if no such attribute exists in the entry, returns NULL and sets the appropriate error code in the LDAP structure. To get the error code, call the `ldap_get_lderrno()` function.

See Chapter 18, “Result Codes” for a complete listing of error codes.

Description

Use the `ldap_get_values()` routine if the attribute values are string values.

For more information, see “Getting the Values of an Attribute” on page 128 of Chapter 5, “Searching a LDAP Directory.”

Example

Code Example 17-26 gets the first value of the `jpegPhoto` attribute and saves the JPEG data to a file.

Code Example 17-26 `ldap_get_values_len()` Code Example

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result, *e;
BerElement *ber;
struct berval photo_data;
struct berval **list_of_photos;
FILE *out;
char *my_searchbase = "dc=example,dc=com";
char *my_filter = "(sn=Jensen)";
...
/* Search the directory */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, my_filter, NULL,
0, &result ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_search_s" );
    return( 1 );
}

/* Get the first matching entry.*/
e = ldap_first_entry( ld, result );

/* Get the value of the jpegPhoto attribute */
if ( ( list_of_photos = ldap_get_values_len( ld, e, "jpegPhoto" ) ) != NULL
) {
    /* Prepare to write the JPEG data to a file */
    if ( ( out = fopen( "photo.jpg", "wb" ) ) == NULL ) {
        perror( "fopen" );
        return( 1 );
    }
}
```

Code Example 17-26 ldap_get_values_len() Code Example (*Continued*)

```

/* Get the first JPEG */
photo_data = *list_of_photos[0];
/* Write the JPEG data to a file */
fwrite( photo_data.bv_val, photo_data.bv_len, 1, out );
fclose( out );
/* Free the attribute values from memory when done. */
ldap_value_free_len( list_of_photos );
}
...

```

See Also

ldap_first_entry(), ldap_first_attribute(), ldap_get_values(),
 ldap_next_entry(), ldap_next_attribute()

ldap_init()

The ldap_init() function initializes a session with an LDAP server and returns an LDAP structure that represents the context of the connection to that server.

Syntax

```

#include <ldap.h>
LDAP * ldap_init( const char *defhost, int defport );

```

Parameters

This function has the parameters listed in Table 17-110.

Table 17-110ldap_init() Function Parameters

Parameter	Description
defhost	Space-delimited list of one or more host names (or IP address in dotted notation, such as "141.211.83.36") of the LDAP servers that you want the LDAP client to connect to. The names can be in <i>hostname:portnumber</i> format (in which case, <i>portnumber</i> overrides the port number specified by the defport argument.
defport	Default port number of the LDAP server. To specify the standard LDAP port (port 389), use LDAP_PORT as the value for this parameter.

Returns

One of the following values:

- If successful, returns a pointer to an LDAP structure.
- If unsuccessful, returns a NULL.

Description

The `ldap_init()` function initializes a session with an LDAP server by allocating an LDAP structure containing information about the session including the host name and port of the LDAP server, preferences for the session (such as the maximum number of entries to return in a search), and the error code of the last LDAP operation performed. (The LDAP structure defaults to LDAPv2. It is highly recommended to set it to LDAPv3 using the `ldap_set_option()`.)

CAUTION `ldap_init()` does not open a connection to the LDAP server. The actual connection opening will occur when the first operation is attempted. Certain fields in the LDAP structure can be set using `ldap_set_option()`.

You can specify a list of LDAP servers to which you want to attempt to connect by passing a space-delimited list of the host names as the `defhost` argument. Your client will attempt to connect to the first LDAP server in the list. If the attempt fails, your client will attempt to connect to the next LDAP server in the list. In Code Example 17-27, the LDAP client will attempt to connect to the LDAP server on `ld1.example.com`, port 389. If that server does not respond, the client will attempt to connect to the LDAP server on `ld2.example.com`, port 389. If that server does not respond, the client will use the server on `ld3.example.com`, port 389.

Code Example 17-27 Space-Delimited List for `ldap_init()`

```
...
LDAP *ld
...
ld = ldap_init( "ld1.example.com ld2.example.com \
ld3.example.com", LDAP_PORT );
```

If any of the servers do not use the default port specified by the `defport` argument, use the *hostname:portnumber* format to specify the server name. In Code Example 17-28, the LDAP client will attempt to connect to the LDAP server on `ld1.example.com`, port 389. If that server does not respond, the client will attempt to connect to the LDAP server on `ld2.example.com`, port 38900.

Code Example 17-28 ldap_init() Example Using Variable Argument

```

...
LDAP *ld
...
ld = ldap_init( "ld1.example.com ld2.example.com:38900", \ LDAP_PORT );

```

For more information, see “Initializing an LDAP Session” on page 61 of Chapter 3, “Writing an LDAP Client.”

NOTE If you are connecting to a secure LDAP server over SSL, you should be calling the ldapssl_init() function. For details, see Chapter 11, “Connecting Over SSL.”

Example

Code Example 17-29 initializes a session with the LDAP server at ldap.example.com:389, and sets a session preference that identifies the client as an LDAPv3 client.

Code Example 17-29 ldap_init() Code Example

```

#include <ldap.h>
...
LDAP *ld;

/* Specify the host name of the LDAP server. */
char *ldap_host = "ldap.example.com";

/* Because the LDAP server is running on the standard LDAP port (port 389),
you can use LDAP_PORT to identify the port number. */
int ldap_port = LDAP_PORT;
...
/* Initialize the session with ldap.example.com:389 */
if ( ( ld = ldap_init( ldap_host, ldap_port ) ) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
}

/* Specify the LDAP version supported by the client. */
version = LDAP_VERSION3;
ldap_set_option( ld, LDAP_OPT_PROTOCOL_VERSION, &version );

...
/* Subsequent calls that authenticate to the LDAP server. */
...

```

See Also

ldap_unbind(), ldap_unbind_s(), ldap_simple_bind(), ldap_simple_bind_s()

ldap_init_getfilter()

The ldap_init_getfilter() function reads a valid LDAP filter configuration file (such as ldapfilter.conf) and returns a pointer to an LDAPFiltDesc structure.

NOTE

ldapfilter.conf, the sample LDAP filter configuration file located in the *DSRK_base/lib/ldapsdk/etc* directory, can be used in context with ldap_init_getfilter().

Syntax

```
#include <ldap.h>
LDAPFiltDesc * ldap_init_getfilter( char *fname );
```

Parameters

This function has the parameters listed in Table 17-111.

Table 17-111ldap_init_getfilter() Function Parameter

Parameter	Description
fname	Name of the LDAP filter configuration file to use.

Returns

One of the following values:

- If successful, returns a pointer to an LDAPFiltDesc structure.
- If unsuccessful (for example, if there is an error reading the file), returns a NULL.

Description

For more information, see “Loading a Filter Configuration File” on page 143 of Chapter 6, “Using Filter Configuration Files.”

Example

Code Example 17-30 loads a filter configuration file named myfilters.conf into memory.

Code Example 17-30 Loading a Filter Configuration File

```
#include <ldap.h>
...
LDAPFiltDesc *lfdp;
char *filter_file = "myfilters.conf";
...
lfdp = ldap_init_getfilter( filter_file );
...
```

See Also

ldap_init_getfilter_buf(), ldap_getfilter_free()

ldap_init_getfilter_buf()

The ldap_init_getfilter_buf() function reads LDAP filter configuration information from a buffer and returns a pointer to an LDAPFiltDesc structure.

Syntax

```
#include <ldap.h>
LDAPFiltDesc * ldap_init_getfilter_buf(char *buf, long buflen );
```

Parameters

This function has the parameters listed in Table 17-112.

Table 17-112ldap_init_getfilter_buf() Function Parameters

Parameter	Description
buf	Buffer containing LDAP filter configuration information.
buflen	Size of the buffer

Returns

One of the following values:

- If successful, returns a pointer to an LDAPFiltDesc structure.
- If unsuccessful (for example, if there is an error reading the file), returns a NULL.

Description

The configuration information in the buffer must use the correct syntax, as specified in “The Filter Configuration File Syntax” on page 140 of Chapter 6, “Using Filter Configuration Files.”

Example

Code Example 17-31 copies the following filter configuration to a buffer in memory and uses this buffer to fill an LDAPFiltDesc structure.

"ldap-example"

```
"@"      " "      "(mail=%v)"      "email address"
          "(mail=%v*)"      "start of email address"
```

Code Example 17-31 ldap_init_getfilter() Code Example

```
#include <string.h>
#include <ldap.h>
...
LDAPFiltDesc *lfdp;
char filtbuf[ 1024 ];
...
/* Create the filter config buffer */
strcpy( filtbuf, "\"ldap-example\"\n" );
strcat( filtbuf, " \"@\" \" \" \"(mail=%v)\" \"\t\" \"email address\"\n" );
strcat( filtbuf, " \"\t\t\" \"(mail=%v*)\" \"\t\" \"start of email address\"\n" );
lfdp = ldap_init_getfilter( filtbuf, strlen( filtbuf ) );
...
```

See Also

ldap_init_getfilter(), ldap_getfilter_free()

ldap_init_searchprefs()

The ldap_init_searchprefs() function reads a sequence of search preference configurations from a valid LDAP searchpref configuration file.

Syntax

```
#include <ldap.h>
int ldap_init_searchprefs( char *file, struct ldap_searchobj **solistp );
```

Parameters

This function has the parameters listed in Table 17-113.

Table 17-113 ldap_init_searchprefs() Function Parameters

Parameter	Description
file	Pointer to a valid LDAP searchpref configuration file.
solistp	Pointer to a list of search preference data structures.

Returns

One of the following values:

- LDAP_SEARCHPREF_ERR_VERSION.
- LDAP_SEARCHPREF_ERR_MEM if there is a memory allocation problem.
- Upon success, 0 is returned and solistp is set to point to a list of search preference data structures.

Description

ldap_init_searchprefs() provides a standard way to access LDAP search preference configuration data.

NOTE

LDAP search preference configurations are typically used by LDAP client programs to specify which attributes a user may search by, labels for the attributes, and LDAP filters and scopes associated with those searches. Client software presents these choices to a user, who can then specify the type of search to be performed. See the man page for ldapsearchprefs.conf for more information.

See Also

ldap_free_searchprefs(), ldap_init_searchprefs_buf()

ldap_init_searchprefs_buf()

The ldap_init_searchprefs_buf() function reads a sequence of search preference configurations from the parameter, buf.

Syntax

```
#include <ldap.h>
int ldap_init_searchprefs_buf( char *buf, long buflen,
    struct ldap_searchobj **solistp );
```

Parameters

This function has the parameters listed in Table 17-114.

Table 17-114ldap_init_searchprefs_buf() Function Parameters

Parameter	Description
buf	Pointer to data in the format defined for an LDAP search preference configuration file.
buflen	The size of buf.
solistp	Pointer to a list of search preference data structures.

Returns

One of the following values:

- LDAP_SEARCHPREF_ERR_VERSION if buf points to data that is newer than can be handled.
- LDAP_SEARCHPREF_ERR_MEM if there is a memory allocation problem.
- Upon success, 0 is returned and solistp is set to point to a list of search preference data structures.

Description

ldap_init_searchprefs_buf() reads a sequence of search preference configurations from buf.

NOTE	LDAP search preference configurations are typically used by LDAP client programs to specify which attributes a user may search by, labels for the attributes, and LDAP filters and scopes associated with those searches. Client software presents these choices to a user, who can then specify the type of search to be performed. See the man page for ldapsearchprefs.conf for more information.

See Also

ldap_init_searchprefs(), ldap_free_searchprefs()

ldap_init_templates()

The ldap_init_templates() function reads a sequence of templates from a valid LDAP template configuration file.

Syntax

```
#include <disptmpl.h>
int ldap_init_templates( char *file, struct ldap_disptmpl **tmplist );
```

Parameters

This function has the parameters listed in Table 17-115.

Table 17-115 ldap_init_templates() Function Parameters

Parameter	Description
file	Pointer to a valid LDAP template configuration file.
tmplist	Pointer to a list of template data structures.

Returns

- If successful, 0 is returned and tmplist is configured.
- Upon error:
 - LDAP_TMPL_ERR_VERSION if buf points to data that is newer than can be handled.
 - LDAP_TMPL_ERR_MEM if there is a memory allocation problem.
 - LDAP_TMPL_ERR_SYNTAX if there is a problem with the format of the templates buffer or file.
 - LDAP_TMPL_ERR_FILE if the file cannot be read.

Description

ldap_init_templates() reads a sequence of templates from a valid LDAP template configuration file. Each template defined in the file is an ldap_disptmpl structure.

See Also

ldap_init_templates_buf(), ldap_free_templates()

ldap_init_templates_buf()

The ldap_init_templates_buf() function reads a sequence of templates from a buffer.

Syntax

```
#include <disptmpl.h>
int ldap_init_templates_buf( char *buf, long buflen,
    struct ldap_disptmpl **tmplist );
```

Parameters

This function has the parameters listed in Table 17-116.

Table 17-116ldap_init_templates_buf() Function Parameters

Parameter	Description
buf	Pointer to data in the format defined for a valid LDAP template configuration file.
buflen	The size of buf.
tmplist	Pointer to a list of template data structures.

Returns

- If successful, 0 is returned and tmplist is configured.
- Upon error:
 - LDAP_TMPL_ERR_VERSION if buf points to data that is newer than can be handled.
 - LDAP_TMPL_ERR_MEM if there is a memory allocation problem.
 - LDAP_TMPL_ERR_SYNTAX if there is a problem with the format of the templates buffer or file.
 - LDAP_TMPL_ERR_FILE if the file cannot be read.

Description

The ldap_init_templates_buf() function reads a sequence of templates from a buffer.

See Also

ldap_init_templates(), ldap_free_templates()

ldap_is_dns_dn()

CAUTION This function is deprecated and should not be used. It is included in ldap-to-be-deprecated.h for backward-compatibility.

The `ldap_is_dns_dn()` function determines whether a DN string is of the experimental DNS-style DN (generally in the form of an RFC 822 e-mail address) or the RFC 1779 formatted DN.

Description

This function is specific to LDAPv2 and should not be used when dealing with LDAPv3 servers or data.

NOTE

More information can be found in *RFC 1779 - A String Representation of Distinguished Names* (<http://www.faqs.org/rfcs/rfc1779.html>) and *RFC 822 - Standard for the Format of ARPA Internet Text Messages* (<http://www.faqs.org/rfcs/rfc822.html>).

ldap_is_ldap_url()

The `ldap_is_ldap_url()` function determines whether or not a URL is an LDAP URL.

Syntax

```
#include <ldap.h>
int ldap_is_ldap_url( const char *url );
```

Parameters

This function has the parameter listed in Table 17-117.

Table 17-117 ldap_is_ldap_url() Function Parameter

Parameter	Description
url	The URL that you want to check.

Returns

One of the following values:

- 1 if the URL is an LDAP URL.
- 0 if the URL is not an LDAP URL.

Description

The `ldap_is_ldap_url()` function determines whether or not a URL is an LDAP URL. An LDAP URL is a URL with the protocol set to `ldap://` (or `ldaps://`, if the server is communicating over a SSL connection).

For more information, see “Determining an LDAP URL” on page 202 of Chapter 9, “Working with LDAP URLs.”

Example

Code Example 17-32 determines if a URL is a LDAP URL.

Code Example 17-32 ldap_is_ldap_url() Code Example

```
#include <stdio.h>
#include <ldap.h>
...
char *my_url = "ldap://ldap.sun.com/dc=example,dc=com";
...
if ( ldap_is_ldap_url( my_url ) != 0 ) {
    printf( "%s is an LDAP URL.\n", my_url );
} else {
    printf( "%s is not an LDAP URL.\n", my_url );
}
...
```

See Also

ldap_url_parse()

ldap_keysort_entries()

The ldap_keysort_entries() function is used for sorting entries.

Syntax

```
#include <ldap-extension.h>
int ldap_keysort_entries( LDAP *ld, LDAPMessage **chain,
    void *arg, LDAP_KEYGEN_CALLBACK *gen, LDAP_KEYCMP_CALLBACK *cmp,
    LDAP_KEYFREE_CALLBACK *fre );
```

Parameters

This function has the parameters listed in Table 17-118.

Table 17-118 ldap_keysort_entries() Function Parameter

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
chain	Chain of entries returned by the ldap_result() or ldap_search_s() function.

Table 17-118 ldap_keysort_entries() Function Parameter

Parameter	Description
arg	Pointer to an additional argument that you want to pass.
gen	Callback used to generate the key(s) for sorting once the compare function has been applied.
cmp	Comparison function used when sorting the values. For details, see “Sorting Entries by an Attribute” on page 132 of Chapter 5, “Searching a LDAP Directory.”
fre	Callback used to free the key once the compare function has been applied.

See Also

LDAP_KEYGEN_CALLBACK, LDAP_KEYCMP_CALLBACK, LDAP_KEYFREE_CALLBACK

ldap_memcache_destroy()

The `ldap_memcache_destroy()` function frees an `LDAPMemCache` structure from memory.

Syntax

```
#include <ldap.h>
void ldap_memcache_destroy( LDAPMemCache *cache );
```

Parameters

This function has the parameter listed in Table 17-119.

Table 17-119 ldap_memcache_destroy() Function Parameter

Parameter	Description
cache	Pointer to the <code>LDAPMemCache</code> structure that you want freed from memory.

Description

The `ldap_memcache_destroy()` function frees the specified `LDAPMemCache` structure from memory. Call this function after you are done working with a cache.

See Also

`ldap_memcache_init()`

ldap_memcache_flush()

The ldap_memcache_flush() function flushes items from the specified cache.

Syntax

```
#include <ldap.h>
void ldap_memcache_flush( LDAPMemCache *cache, char *dn,
    int scope );
```

Parameters

This function has the parameters listed in Table 17-120.

Table 17-120 ldap_memcache_flush() Function Parameters

Parameter	Description
cache	Pointer to the LDAPMemCache structure that you want to flush entries from.
dn	Base DN identifying the search requests that you want flushed from the cache. If the base DN of a search request is within the scope specified by this DN and the scope argument, the search request is flushed from the cache. If this argument is NULL, the entire cache is flushed.
scope	Scope that (together with the dn argument) identifies the search requests that you want flushed from the cache. If the base DN of the request is within the scope specified by this argument and the dn argument, the request is flushed from the cache. This argument can have one of the following values: - LDAP_SCOPE_BASE - LDAP_SCOPE_ONELEVEL - LDAP_SCOPE_SUBTREE

Description

The ldap_memcache_flush() function flushes search requests from the cache. If the base DN of a search request is within the scope specified by the dn and scope arguments, the search request is flushed from the cache. If no DN is specified, the entire cache is flushed.

ldap_memcache_get()

The ldap_memcache_get() function gets the in-memory cache associated with an LDAP connection handle.

Syntax

```
#include <ldap.h>
int ldap_memcache_get( LDAP *ld, LDAPMemCache **cachep );
```

Parameters

This function has the parameters listed in Table 17-121.

Table 17-121 ldap_memcache_get() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
cachep	When you call ldap_memcache_get(), it sets this parameter to the pointer to the LDAPMemCache structure associated with the connection handle.

Returns

One of the following values:

- LDAP_SUCCESS if the cache for the specified connection handle was retrieved successfully.
- LDAP_PARAM_ERROR if an invalid parameter was passed to the function.

Description

The ldap_memcache_get() function gets the cache associated with the specified connection handle (LDAP structure). This cache is used by all search requests made through that connection. You can call this function if you want to associate a cache with multiple LDAP connection handles. For example, you can call this function to get the cache associated with one connection, then you can call the ldap_memcache_set() function to associate the cache with another connection.

See Also

ldap_memcache_set()

ldap_memcache_init()

The ldap_memcache_init() function creates an in-memory cache for your LDAP client that you can associate with an LDAP connection.

Syntax

```
#include <ldap.h>
int ldap_memcache_init( unsigned long ttl, unsigned long size,
    char **baseDNs, struct ldap_thread_fns *thread_fns,
    LDAPMemCache **cachep );
```

Parameters

This function has the parameters listed in Table 17-122.

Table 17-122 ldap_memcache_init() Function Parameters

Parameter	Description
ttl	The maximum amount of time (in seconds) that an item can be cached. If 0, there is no limit to the amount of time that an item can be cached.
size	Maximum amount of memory (in bytes) that the cache will consume. If 0, the cache has no size limit.
baseDNs	An array of the base DN strings representing the base DN's of the search requests you want cached. If not NULL, only the search requests with the specified base DN's will be cached. If NULL, all search requests are cached.
thread_fns	An ldap_thread_fns structure specifying the functions that you want used to ensure that the cache is thread-safe. You should specify this if you have multiple threads that are using the same connection handle and cache. If you are not using multiple threads, pass NULL for this parameter.
cachep	When you call this function, it sets this parameter to the pointer to the newly created LDAPMemCache structure.

Returns

One of the following values:

- LDAP_SUCCESS if the cache for the specified connection handle was retrieved successfully.
- LDAP_PARAM_ERROR if an invalid parameter was passed to the function.
- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_SIZELIMIT_EXCEEDED if the initial size of the cache (specified by the size argument) is too small.

Description

The `ldap_memcache_init()` function creates an in-memory, client-side cache that you can use to cache search requests. The function passes back a pointer to an `LDAPMemCache` structure, which represents the cache. You should call the `ldap_memcache_set()` function to associate this cache with an LDAP connection handle (an LDAP structure).

The cache uses search criteria as the key to cached items. When you send a search request, the cache checks the search criteria to determine if that request has been cached previously. If the request was cached, the search results are read from the cache. To flush the cache, call the `ldap_memcache_flush()` function. When you are done with the cache, you can free it from memory by calling the `ldap_memcache_destroy()` function.

CAUTION On Windows® NT, if the `ldap_memcache_init()` function returns an `LDAP_PARAM_ERROR` result code, verify that your client is using the version of the `nsldap32v30.dll` file provided with the LDAP SDK for C, v5.2.

See Also

`LDAPMemCache`, `ldap_memcache_set()`, `ldap_memcache_flush()`, `ldap_memcache_destroy()`, `ldap_memcache_update()`

ldap_memcache_set()

The `ldap_memcache_set()` function associates an in-memory cache with an LDAP connection handle.

Syntax

```
#include <ldap.h>
int ldap_memcache_set( LDAP *ld, LDAPMemCache *cache );
```

Parameters

This function has the parameters listed in Table 17-123.

Table 17-123 `ldap_memcache_set()` Function Parameters

Parameter	Description
<code>ld</code>	Pointer to an LDAP structure containing information about the connection to the LDAP server.
<code>cache</code>	Pointer to an <code>LDAPMemCache</code> structure, which represents the cache that you want used for this connection.

Returns

One of the following values:

- LDAP_SUCCESS if the cache for the specified connection handle was retrieved successfully.
- LDAP_PARAM_ERROR if an invalid parameter was passed to the function.
- LDAP_SIZELIMIT_EXCEEDED if the initial size of the cache (specified by the size argument) is too small.

Description

The ldap_memcache_set() function associates a cache (created by calling ldap_memcache_init()) with an LDAP connection handle. You can call this function if you want to associate a cache with multiple LDAP connection handles. For example, you can call the ldap_memcache_get() function to get the cache associated with one connection, then you can call ldap_memcache_set() to associate the cache with another connection.

After you call this function, search requests made over the specified LDAP connection will use this cache. Calling the ldap_unbind() function will disassociate the cache from the LDAP connection handle.

See Also

ldap_memcache_init(), ldap_memcache_get()

ldap_memcache_update()

The ldap_memcache_update() function checks the cache for items that have expired and removes them.

Syntax

```
#include <ldap.h>
void ldap_memcache_update( LDAPMemCache *cache );
```

Parameters

This function has the parameter listed in Table 17-124.

Table 17-124 ldap_memcache_update() Function Parameter

Parameter	Description
cache	Pointer to an LDAPMemCache structure, which represents the cache that you want to updated.

Description

The `ldap_memcache_update()` function checks the cache for items that have expired and removes them. This check is typically done as part of the way the cache normally works. You do not need to call this function unless you want to update the cache at this point in time.

TIP This function is only useful in a multithreaded application, since it will not return until the cache is destroyed.

See Also

`ldap_memcache_flush()`

ldap_memfree()

The `ldap_memfree()` function frees memory allocated by an LDAP API function call.

Syntax

```
#include <ldap.h>
void ldap_memfree( void *p );
```

Parameters

This function has the parameter listed in Table 17-125.

Table 17-125 ldap_memfree() Function Parameter

Parameter	Description
p	Pointer to memory used by the LDAP library.

Description

For more information, see “Managing Memory” on page 86 of Chapter 4, “Using the LDAP API.”

Example

Code Example 17-33 frees the memory allocated by the `ldap_get_dn()` function.

Code Example 17-33 ldap_memfree() Code Example

```
#include <ldap.h>
...
LDAP *ld;
char *dn;
LDAPMessage *entry;
...
/* Get the distinguished name (DN) for an entry */
dn = ldap_get_dn( ld, entry );
...
/* When you are finished working with the DN, free the memory allocated for
the DN */
ldap_memfree( dn );
...
```

See Also

ldap_free_friendlymap(), ldap_free_urldesc(), ldap_msgfree(),
ldap_value_free(), ldap_value_free_len()

ldap_modify()

The ldap_modify() function modifies an existing entry in the directory asynchronously.

NOTE This is an older function included in the SDK for backward-compatibility. If you are writing a new LDAP client, use ldap_modify_ext() instead.

Syntax

```
#include <ldap.h>
int ldap_modify( LDAP *ld, const char *dn, LDAPMod **mods );
```

Parameters

This function has the parameters listed in Table 17-126.

Table 17-126 ldap_modify() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.

Table 17-126 ldap_modify() Function Parameters (*Continued*)

Parameter	Description
dn	DN of the entry to modify.
mods	Pointer to a NULL-terminated array of pointers to LDAPMod structures representing the attributes that you want to modify.

Returns

Returns the message ID of the ldap_modify() operation. To check the result of this operation, call ldap_result() and ldap_result2error(). For a list of possible result codes for an LDAP modify operation, see ldap_modify_ext_s().

Description

Please use the newer version of this function, ldap_modify_ext().

Example

Code Example 17-34 uses the asynchronous ldap_modify() function to modify the entry for *Barbara Jensen* in the directory. It makes the following changes to the entry:

- Adds the homePhone attribute.
- Changes the telephoneNumber attribute.
- Removes the facsimileTelephoneNumber attribute.

Code Example 17-34 ldap_modify() Code Example

```
#include <ldap.h>
...
LDAP *ld;
LDAPMod *list_of_attrs[4];
LDAPMod attribute1, attribute2, attribute3;
LDAPMessage *result;
int msgid, rc;
struct timeval tv;

/* Distinguished name of the entry that you want to modify. */
char *dn = "uid=bjensen,ou=People,dc=example,dc=com";

/* Values to add or change */
char *homePhone_values[] = { "555-1212", NULL };
char *telephoneNumber_values[] = { "869-5309", NULL };

...
/* Specify each change in separate LDAPMod structures */
attribute1.mod_type = "homePhone";
attribute1.mod_op = LDAP_MOD_ADD;
```

Code Example 17-34 ldap_modify() Code Example (*Continued*)

```

attribute1.mod_values = homePhone_values;
attribute2.mod_type = "telephoneNumber";
attribute2.mod_op = LDAP_MOD_REPLACE;
attribute2.mod_values = telephoneNumber_values;
attribute3.mod_type = "facsimileTelephoneNumber";
attribute3.mod_op = LDAP_MOD_DELETE;
attribute3.mod_values = NULL;
/* NOTE: When removing entire attributes, you need to specify a NULL value
for the mod_values field. */

/* Add the pointers to these LDAPMod structures to an array */
list_of_attrs[0] = &attribute1;
list_of_attrs[1] = &attribute2;
list_of_attrs[2] = &attribute3;
list_of_attrs[3] = NULL;
...
/* Set up the timeout period to wait for the "modify" operation */
tv.tv_sec = tv.tv_usec = 0;

/* Change the entry */
if ( ( msgid = ldap_modify( ld, dn, list_of_attrs ) ) == -1 ) {
    ldap_perror( ld, "ldap_modify" );
    return( 1 );
}

/* Check to see if the operation has completed */
while ( ( rc = ldap_result( ld, msgid, 0, &tv, &result ) ) == 0 ) {
    ...
    /* do other work while waiting for the operation to complete */
    ...
}

/* Check the result to see if any errors occurred */
ldap_result2error( ld, result, 1 );
ldap_perror( ld, "ldap_modify" );
...

```

See Also

ldap_modify_ext()

ldap_modify_ext()

The `ldap_modify_ext()` function modifies an existing entry in the directory asynchronously.

NOTE ldap_modify_ext() is a new version of the ldap_modify() function. If you are writing a new LDAP client, use ldap_modify_ext().

Syntax

```
#include <ldap.h>
int ldap_modify_ext( LDAP *ld, const char *dn, LDAPMod **mods,
    LDAPControl **serverctrls, LDAPControl **clientctrls,
    int *msgidp );
```

Parameters

This function has the parameters listed in Table 17-127.

Table 17-127 ldap_modify_ext() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
dn	DN of the entry to modify.
mods	Pointer to a NULL-terminated array of pointers to LDAPMod structures representing the attributes that you want to modify.
serverctrls	Pointer to an array of LDAPControl structures representing LDAP server controls that apply to this operation. If you do not want to pass any server controls, specify NULL for this argument.
clientctrls	Pointer to an array of LDAPControl structures representing LDAP client controls that apply to this operation. If you do not want to pass any client controls, specify NULL for this argument.
msgidp	Pointer to an integer that will be set to the message ID of the LDAP operation. To check the result of this operation, call ldap_result() and ldap_parse_result().

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if an invalid parameter was passed to the function.
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the control.
- LDAP_SERVER_DOWN if the LDAP server did not receive the request or if the connection to the server was lost.

- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_NOT_SUPPORTED if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3. Make sure that you set the version of your LDAP client to version 3 before calling this function. For details, see “Specifying the LDAP Version of Your Client” on page 65 of Chapter 3, “Writing an LDAP Client.”

Description

The `ldap_modify_ext()` modifies an entry in the directory asynchronous; it does not directly return results. If you want the results to be returned directly by the function, call the synchronous function `ldap_modify_ext_s()` instead. In order to get the results of this LDAP modify operation, you need to call the `ldap_result()` and the `ldap_parse_result()` functions. To make changes to an entry to the directory, you need to specify the following information:

- A unique DN identifying the new entry
Use the `dn` argument to specify the DN of the entry you want to modify.
- A set of attributes for the new entry
Create an `LDAPMod` structure for changes that you want to make to an attribute. Create an array of these `LDAPMod` structures and pass the array as the `mods` argument.

For more information on modifying entries in the directory, see “Modifying an Entry” on page 161 of Chapter 17, “Function Reference.” For a list of possible result codes for an LDAP modify operation, see `ldap_modify_ext_s()`. For information on using the `LDAPMod` structure to specify a change, see “LDAPMod” on page 304 of Chapter 16, “Data Type Reference.”

Example

See Code Example 7-17 on page 170 of Chapter 7, “Adding, Modifying and Deleting Entries.”

See Also

`ldap_modify_ext_s()`, `ldap_result()`, `ldap_parse_result()`, `LDAPMod`

ldap_modify_ext_s()

The `ldap_modify_ext_s()` modifies an existing entry in the directory synchronously.

NOTE `ldap_modify_ext_s()` is a new version of the `ldap_modify_s()` function. If you are writing a new LDAP client, use `ldap_modify_ext_s()`.

Syntax

```
#include <ldap.h>
int ldap_modify_ext_s( LDAP *ld, const char *dn, LDAPMod **mods,
    LDAPControl **serverctrls, LDAPControl **clientctrls );
```

Parameters

This function has the parameters listed in Table 17-128.

Table 17-128 `ldap_modify_ext_s()` Function Parameters

Parameter	Description
<code>ld</code>	Pointer to an LDAP structure containing information about the connection to the LDAP server.
<code>dn</code>	DN of the entry to modify.
<code>mods</code>	Pointer to a NULL-terminated array of pointers to <code>LDAPMod</code> structures representing the attributes that you want to modify.
<code>serverctrls</code>	Pointer to an array of <code>LDAPControl</code> structures representing LDAP server controls that apply to this operation. If you do not want to pass any server controls, specify NULL for this argument.
<code>clientctrls</code>	Pointer to an array of <code>LDAPControl</code> structures representing LDAP client controls that apply to this operation. If you do not want to pass any client controls, specify NULL for this argument.

Returns

One of the following values:

- `LDAP_SUCCESS` if successful.
- `LDAP_PARAM_ERROR` if an invalid parameter was passed to the function.
- `LDAP_ENCODING_ERROR` if an error occurred when BER-encoding the control.
- `LDAP_SERVER_DOWN` if the LDAP server did not receive the request or if the connection to the server was lost.
- `LDAP_NO_MEMORY` if memory cannot be allocated.
- `LDAP_LOCAL_ERROR` if an error occurred when receiving the results from the server.

- `LDAP_DECODING_ERROR` if an error occurred when decoding the BER-encoded results from the server.
- `LDAP_NOT_SUPPORTED` if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3. Make sure that you set the version of your LDAP client to version 3 before calling this function. For details, see “Specifying the LDAP Version of Your Client” on page 65 of Chapter 3, “Writing an LDAP Client.”

NOTE

Sun Java System Directory Server and other LDAP server products may send result codes in addition to those described. For example, the server may have loaded a plug-in that returns custom result codes. Check your LDAP server documentation for other result codes.

Description

The `ldap_modify_ext_s()` modifies an entry in the directory.

`ldap_modify_ext_s()` is a synchronous function, which directly returns the results of the operation. If you want to perform other operations while waiting for the results of this operation, call the asynchronous function `ldap_modify_ext()` instead.

To make changes to an entry to the directory, you need to specify the following information:

- A unique DN identifying the new entry

Use the `dn` argument to specify the DN of the entry that you want to modify.

- A set of attributes for the new entry

Create an `LDAPMod` structure for change that you want to make to an attribute. Create an array of these `LDAPMod` structures and pass the array as the `mods` argument.

For more information on modifying entries in the directory, see “Modifying an Entry” on page 161 of Chapter 17, “Function Reference.” For information on using the `LDAPMod` structure to specify a change, see “`LDAPMod`” on page 304 of Chapter 16, “Data Type Reference.”

Example

See Code Example 7-15 on page 167 of Chapter 7, “Adding, Modifying and Deleting Entries.”

See Also

`ldap_modify_ext()`, `LDAPMod`

ldap_modify_s()

The `ldap_modify_s()` function modifies an existing entry in the directory synchronously.

NOTE

This is an older function that is included in the SDK for backward-compatibility. If you are writing a new LDAP client, use `ldap_modify_ext_s()` instead.

Syntax

```
#include <ldap.h>
int ldap_modify_s( LDAP *ld, const char *dn, LDAPMod **mods );
```

Parameters

This function has the parameters listed in Table 17-129.

Table 17-129 ldap_modify_s() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
dn	DN of the entry to modify.
mods	Pointer to a NULL-terminated array of pointers to LDAPMod structures representing the attributes that you want to modify.

Returns

For a list of possible result codes for an LDAP modify operation, see `ldap_modify_ext_s()`.

Description

Please use the newer version of this function, `ldap_modify_ext_s()`.

Example

Code Example 17-35 uses the synchronous `ldap_modify_s()` function to makes the following changes to the *Barbara Jensen* entry:

- Adds the `homePhone` attribute.
- Changes the `telephoneNumber` attribute.
- Removes the `facsimileTelephoneNumber` attribute.

Code Example 17-35 ldap_modify_s() Code Example

```

#include <ldap.h>
...
LDAP *ld;
LDAPMod *list_of_attrs[4];
LDAPMod attribute1, attribute2, attribute3;
LDAPControl **srvrctrls, **clntctrls;

/* Distinguished name of the entry that you want to modify. */
char *dn = "uid=bjensen,ou=People,dc=example,dc=com";

/* Values to add or change */
char *homePhone_values[] = { "555-1212", NULL };
char *telephoneNumber_values[] = { "869-5309", NULL };

...
/* Specify each change in separate LDAPMod structures */
attribute1.mod_type = "homePhone";
attribute1.mod_op = LDAP_MOD_ADD;
attribute1.mod_values = homePhone_values;
attribute2.mod_type = "telephoneNumber";
attribute2.mod_op = LDAP_MOD_REPLACE;
attribute2.mod_values = telephoneNumber_values;
attribute3.mod_type = "facsimileTelephoneNumber";
attribute3.mod_op = LDAP_MOD_DELETE;
attribute3.mod_values = NULL;
/* NOTE: When removing entire attributes, you need to specify a NULL value
for the mod_values or mod_bvalues field. */

/* Add the pointers to these LDAPMod structures to an array */
list_of_attrs[0] = &attribute1;
list_of_attrs[1] = &attribute2;
list_of_attrs[2] = &attribute3;
list_of_attrs[3] = NULL;

...
/* Change the entry */
if ( ldap_modify_s( ld, dn, list_of_attrs ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_modify_s" );
    return( 1 );
}
...

```

See Also

ldap_modify_ext_s()

ldap_modrdn()

CAUTION This function is deprecated and should not be used. It is included in `ldap-deprecated.h` for backward-compatibility. Please use `ldap_rename()`.

ldap_modrdn_s()

CAUTION This function is deprecated and should not be used. It is included in `ldap-deprecated.h` for backward-compatibility. Please use `ldap_rename_s()`.

ldap_modrdn2()

CAUTION This function is deprecated and should not be used. It is included in `ldap-deprecated.h` for backward-compatibility. Please use `ldap_rename()` instead.

The `ldap_modrdn2()` function changes the relative distinguished name (RDN) of an entry in the directory asynchronously.

Syntax

```
#include <ldap-deprecated.h>
int ldap_modrdn2( LDAP *ld, const char *dn, const char *newrdn,
    int deleteoldrdn );
```

Parameters

This function has the parameters listed in Table 17-130.

Table 17-130 ldap_modrdn2() Function Parameters

Parameter	Description
<code>ld</code>	Pointer to an LDAP structure containing information about the connection to the LDAP server.
<code>dn</code>	DN of the entry to modify.
<code>newrdn</code>	New RDN to assign to the entry.
<code>deleteoldrdn</code>	If this is a non-zero value, the old RDN is not retained as a value in the modified entry. If 0, the old RDN is retained as an attribute in the modified entry.

Returns

Returns the message ID of the `ldap_modrdn2()` operation. To check the result of this operation, call `ldap_result()` and `ldap_result2error()`. For a list of possible result codes, see `ldap_rename()`.

Description

Please use the newer version of this function, `ldap_rename()`.

Example

Code Example 17-36 uses the asynchronous `ldap_modrdn2()` function to change the RDN of an entry from `uid=bjensen` to `uid=babs`. The code removes the existing RDN `bjensen` from the `uid` attribute of the entry.

Code Example 17-36 `ldap_modrdn2()` Code Example

```
#include <ldap-deprecated.h>
...
LDAP *ld;
LDAPMessage *result;
int msgid, rc;
struct timeval tv;

/* Distinguished name of the entry that you want to rename. */
char *dn = "uid=bjensen,ou=People,dc=example,dc=com";

/* New relative distinguished name (RDN) of the entry */
char *rdn = "uid=babs";
...
/* Set up the timeout period to wait for the "modify RDN" operation */
tv.tv_sec = tv.tv_usec = 0;

/* Rename the entry */
if ( ( msgid = ldap_modrdn2( ld, dn, rdn, 1 ) ) == -1 ) {
    ldap_perror( ld, "ldap_modrdn2" );
    return( 1 );
}

/* Check to see if the operation has completed */
while ( ( rc = ldap_result( ld, msgid, 0, &tv, &result ) ) == 0 ) {
    ...
    /* do other work while waiting for the operation to complete */
    ...
}

/* Check the result to see if any errors occurred */
ldap_result2error( ld, result, 1 );
ldap_perror( ld, "ldap_modrdn2" );
...
```


See Also

ldap_rename()

ldap_modrdn2_s()

CAUTION This function is deprecated and should not be used. It is included in ldap-deprecated.h for backward-compatibility. Please use ldap_rename_s() instead.

The ldap_modrdn2_s() function changes the relative distinguished name (RDN) of an entry in the directory synchronously.

Syntax

```
#include <ldap-deprecated.h>
int ldap_modrdn2_s( LDAP *ld, const char *dn,
    const char *newrdn, int deleteoldrdn );
```

Parameters

This function has the parameters listed in Table 17-131.

Table 17-131 ldap_modrdn2_s() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
dn	DN of the entry to modify.
newrdn	New RDN to assign to the entry.
deleteoldrdn	If this is a non-zero value, the old RDN is not retained as a value in the modified entry. If 0, the old RDN is retained as an attribute in the modified entry.

Returns

For a list of possible result codes, see ldap_rename_s().

Description

Please use the newer version of this function, ldap_rename_s().

Example

Code Example 17-37 uses the synchronous `ldap_modrdn2_s()` function to change the RDN of an entry from `uid=bjensen` to `uid=babs`. The code removes the existing RDN `babs` from the `uid` attribute of the entry.

Code Example 17-37 `ldap_modrdn2_s()` Code Example

```
#include <ldap.h>
...
LDAP *ld;

/* Distinguished name of the entry that you want to rename. */
char *dn = "uid=bjensen,ou=People,dc=example,dc=com";

/* New relative distinguished name (RDN) of the entry */
char *rdn = "uid=babs";

...
/* Rename the entry */
if ( ldap_modrdn2_s( ld, dn, rdn, 1 ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_modrdn2_s" );
    return( 1 );
}
...
```

See Also

`ldap_rename_s()`

ldap_mods_free()

The `ldap_mods_free()` function frees the `LDAPMod` structures that you've allocated to add or modify entries.

NOTE You need to call this function only if you've allocated memory for these structures yourself. For more information, see the `ldap-extension.h` header file.

Syntax

```
#include <ldap.h>
void ldap_mods_free( LDAPMod **mods, int freemods );
```

Parameters

This function has the parameters listed in Table 17-132

Table 17-132 ldap_mods_free() Function Parameters

Parameter	Description
mods	Pointer to a NULL-terminated array of pointers to LDAPMod structures.
freemods	If this is a non-zero value, frees the array of pointers as well as the LDAPMod structures. If 0, just frees the LDAPMod structures.

Example

Code Example 17-38 allocates memory for LDAPMod structures and frees them when done.

Code Example 17-38 ldap_mods_free() Code Example

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
char *dn;
int i, msgid;
LDAPMod **mods;

...
/* Construct the array of values to add */
mods = ( LDAPMod ** ) malloc(( NMODS + 1 ) * sizeof( LDAPMod * ));
if ( mods == NULL ) {
    fprintf( stderr, "Cannot allocate memory for mods array\n" );
}
for ( i = 0; i < NMODS; i++ ) {
    if (( mods[ i ] = ( LDAPMod * ) malloc( sizeof( LDAPMod ))) == NULL) {
        fprintf( stderr, "Cannot allocate memory for mods element\n" );
        exit( 1 );
    }
}
...
/* Code for filling the structures goes here. */
...
/* Initiate the add operation */
if (( msgid = ldap_add( ld, dn, mods )) < 0 ) {
    ldap_perror( ld, "ldap_add" );
    ldap_mods_free( mods, 1 );
    return( 1 );
}
...

```

ldap_msgfree()

The ldap_msgfree() function frees the memory allocated for a result by ldap_result() or ldap_search_s().

Syntax

```
#include <ldap.h>
int ldap_msgfree( LDAPMessage *lm );
```

Parameters

This function has the parameter listed in Table 17-133.

Table 17-133 ldap_msgfree() Function Parameter

Parameter	Description
lm	Pointer to the result to be freed from memory.

Returns

One of the following values:

- LDAP_RES_BIND indicates that the LDAPMessage structure contains the result of an LDAP bind operation.
- LDAP_RES_SEARCH_ENTRY indicates that the LDAPMessage structure contains an entry found during an LDAP search operation.
- LDAP_RES_SEARCH_REFERENCE indicates that the LDAPMessage structure contains an LDAPv3 search reference (a referral to another LDAP server) found during an LDAP search operation.
- LDAP_RES_SEARCH_RESULT indicates that the LDAPMessage structure contains the result of an LDAP search operation.
- LDAP_RES_MODIFY indicates that the LDAPMessage structure contains the result of an LDAP modify operation.
- LDAP_RES_ADD indicates that the LDAPMessage structure contains the result of an LDAP add operation.
- LDAP_RES_DELETE indicates that the LDAPMessage structure contains the result of an LDAP delete operation.
- LDAP_RES_MODRDN or LDAP_RES_RENAME indicates that the LDAPMessage structure contains the result of an LDAP modify DN operation.

- LDAP_RES_COMPARE indicates that the LDAPMessage structure contains the result of an LDAP compare operation.
- LDAP_RES_EXTENDED indicates that the LDAPMessage structure contains the result of an LDAPv3 extended operation.
- LDAP_SUCCESS if the operation times out.
- -1 indicates that the `lm` argument is not a pointer to a valid LDAPMessage structure.
- If unsuccessful, returns the LDAP error code for the operation.

See Chapter 18, “Result Codes” for a complete listing.

Description

For more information, see “Freeing the Search Results” on page 134 of Chapter 5, “Searching a LDAP Directory.”

Example

Code Example 17-39 frees the results of a search.

Code Example 17-39 ldap_msgfree() Code Example

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result;
char *my_searchbase = "dc=example,dc=com";
char *my_filter = "(sn=Jensen)";
char *get_attr[] = { "cn", "mail", NULL };
...
/* Search the directory */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, my_filter,
get_attr, 0, &result ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_search_s" );
    return( 1 );
}
...
/* Free the results when done */
ldap_msgfree( result );
...
```

See Also

ldap_result(), ldap_search_s()

ldap_msgid()

The `ldap_msgid()` function determines the message ID of a result obtained by calling `ldap_result()` or `ldap_search_s()`.

Syntax

```
#include <ldap.h>
int ldap_msgid( LDAPMessage *lm );
```

Parameters

This function has the parameter listed in Table 17-134.

Table 17-134 ldap_msgid() Function Parameter

Parameter	Description
lm	Pointer to the result to check.

Returns

One of the following values:

- The message ID if successful.
- -1 if unsuccessful.

Example

Code Example 17-40 prints the message ID from the result obtained from a synchronous LDAP search operation.

Code Example 17-40 ldap_msgid() Code Example

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result;
...
/* Perform a search */
if ( ldap_search_s( ld, MY_SEARCHBASE, LDAP_SCOPE_SUBTREE, MY_FILTER,
    NULL, 0, &result ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_search_s" );
    return( 1 );
}

/* Get and print the message ID */
if ( ldap_msgid( result ) != -1 ) {
    printf( "Message ID: %d\n" );
}
```

Code Example 17-40 ldap_msgid() Code Example

```

} else {
    printf( "An error occurred.\n" );
}
...

```

See Also

ldap_msgtype(), ldap_result(), ldap_search_s()

ldap_msgtype()

The ldap_msgtype() function determines the type of result obtained by calling ldap_result() or ldap_search_s().

Syntax

```

#include <ldap.h>
int ldap_msgtype( LDAPMessage *lm );

```

Parameters

This function has the parameter listed in Table 17-135.

Table 17-135 ldap_msgtype() Function Parameter

Parameter	Description
lm	Pointer to the LDAPMessage structure to check.

Returns

One of the following values:

- LDAP_RES_BIND indicates that the LDAPMessage structure contains the result of an LDAP bind operation.
- LDAP_RES_SEARCH_ENTRY indicates that the LDAPMessage structure contains an entry found during an LDAP search operation.
- LDAP_RES_SEARCH_REFERENCE indicates that the LDAPMessage structure contains an LDAPv3 search reference (a referral to another LDAP server) found during an LDAP search operation.

- LDAP_RES_SEARCH_RESULT indicates that the LDAPMessage structure contains the result of an LDAP search operation.
- LDAP_RES_MODIFY indicates that the LDAPMessage structure contains the result of an LDAP modify operation.
- LDAP_RES_ADD indicates that the LDAPMessage structure contains the result of an LDAP add operation.
- LDAP_RES_DELETE indicates that the LDAPMessage structure contains the result of an LDAP delete operation.
- LDAP_RES_MODRDN or LDAP_RES_RENAME indicates that the LDAPMessage structure contains the result of an LDAP modify DN operation.
- LDAP_RES_COMPARE indicates that the LDAPMessage structure contains the result of an LDAP compare operation.
- LDAP_RES_EXTENDED indicates that the LDAPMessage structure contains the result of an LDAPv3 extended operation.
- -1 indicates that the `lm` argument is not a pointer to a valid LDAPMessage structure.

Example

Code Example 17-41 prints the message type for a result obtained from a synchronous LDAP search operation.

Code Example 17-41 ldap_msgtype() Code Example

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result;
int msgtype;
...
/* Perform a search */
if ( ldap_search_s( ld, MY_SEARCHBASE, LDAP_SCOPE_SUBTREE, MY_FILTER,
    NULL, 0, &result ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_search_s" );
    return( 1 );
}

/* Get and print the message type */
msgtype = ldap_msgtype( result );
if ( msgtype != -1 ) {
    printf( "Message type: %d\n", msgtype );
} else {
    printf( "An error occurred.\n" );
}
```


Code Example 17-41 ldap_msgtype() Code Example

```
}
...
```

See Also

ldap_msgid(), ldap_result(), ldap_search_s()

ldap_multisort_entries()

The `ldap_multisort_entries()` function sorts a chain of entries retrieved from an LDAP search call (`ldap_search_s()` or `ldap_result()`) by either a specified set of attributes in the entries or DN.

Syntax

```
#include <ldap.h>
int ldap_multisort_entries( LDAP *ld, LDAPMessage **chain,
    char **attr, LDAP_CMP_CALLBACK *cmp );
```

Parameters

This function has the parameters listed in Table 17-136.

Table 17-136 ldap_multisort_entries() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
chain	Chain of entries returned by the <code>ldap_result()</code> or <code>ldap_search_s()</code> function.
attr	Array of attributes to use for sorting the results. If NULL, results are sorted by DN.
cmp	Comparison function used when sorting the values. For details, see “Sorting Entries by an Attribute” on page 132 of Chapter 5, “Searching a LDAP Directory.”

Returns

One of the following values:

- 0 if successful.

- -1 if memory cannot be allocated by this function. (The error code LDAP_NO_MEMORY is set in the LDAP structure. To get the error code, call the ldap_get_lderrno() function.)
- LDAP_PARAM_ERROR if an invalid parameter was passed to the function

Example

Code Example 17-42 sorts entries first by the roomNumber attribute, then by the telephoneNumber attribute.

Code Example 17-42 ldap_multisort_entries() Code Example

```
#include <stdio.h>
#include <string.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result;
char *my_searchbase = "dc=example,dc=com";
char *my_filter = "(sn=Jensen)";
char *attrs[3];
attrs[0] = "roomNumber";
attrs[1] = "telephoneNumber";
attrs[2] = NULL;
...
/* Search the directory */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, my_filter, NULL,
0, &result ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_search_s" );
    return( 1 );
}

/* Sort the results, using strcasecmp */
if ( ldap_multisort_entries( ld, &result, attrs, strcasecmp ) !=
LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_multisort_entries" );
    return( 1 );
}
...
```

See Also

ldap_result(), ldap_search_s(), ldap_sort_entries(), LDAP_CMP_CALLBACK

ldap_name2template()

The ldap_name2template() function obtains a pointer to the correct ldap_disptmpl structure.

Syntax

```
#include <disptmpl.h>
struct ldap_disptmpl * ldap_name2template( char *name
      struct ldap_disptmpl *tmplist );
```

Parameters

This function has the parameters listed in Table 17-137.

Table 17-137 ldap_name2template() Function Parameters

Parameter	Description
name	Name of the template.
tmplist	Pointer to a list of template data structures.

Returns

- If successful, 0 is returned and tmplist is configured.
- Upon error:
 - LDAP_TMPL_ERR_VERSION if buf points to data that is newer than can be handled.
 - LDAP_TMPL_ERR_MEM if there is a memory allocation problem.
 - LDAP_TMPL_ERR_SYNTAX if there is a problem with the format of the templates buffer or file.
 - LDAP_TMPL_ERR_FILE if the file cannot be read.

Description

ldap_name2template() obtains a pointer to the correct ldap_disptmpl structure. Links to templates can be defined by name.

ldap_next_attribute()

The ldap_next_attribute() function returns the name of the next attribute in an entry returned by ldap_first_entry() or ldap_next_entry().

Syntax

```
#include <ldap.h>
char * ldap_next_attribute( LDAP *ld, LDAPMessage *entry,
    BerElement *ber);
```

Parameters

This function has the parameters listed in Table 17-138.

Table 17-138 ldap_next_attribute() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
entry	Pointer to the LDAPMessage structure representing the entry returned by the ldap_first_entry() or ldap_next_entry() function.
ber	A pointer to a BerElement allocated to keep track of its current position. Pass this pointer to subsequent calls to ldap_next_attribute() to step through the entry's attributes.

Returns

One of the following values:

- If successful, returns the name of the next attribute in an entry. When you are done using this data, you should free the memory by calling the ldap_memfree() function.
- If no more attributes exist in the entry, returns a NULL.
- If unsuccessful, returns a NULL and sets the appropriate error code in the LDAP structure. To get the error code, call the ldap_get_lderrno() function.

See Chapter 18, “Result Codes” for a complete listing of error codes.

Description

The ldap_first_attribute() function returns a pointer to a BerElement. You use this pointer with ldap_next_attribute() to iterate through the list of elements. After the last call to ldap_next_element(), you should free the BerElement pointer using ldap_ber_free(). When calling ldap_ber_free(), make sure to specify that the buffer is not freed (pass 0 for the freebuf parameter).

For more information, see “Getting Attribute Names from an Entry” on page 127 of Chapter 5, “Searching a LDAP Directory.”

Example

See the example under `ldap_first_attribute()`.

See Also

`ldap_first_attribute()`, `ldap_getfirstfilter()`, `ldap_next_entry()`

ldap_next_disptmpl()

The `ldap_next_disptmpl()` function returns the next template in a list.

Syntax

```
#include <disptmpl.h>
struct ldap_disptmpl * ldap_next_disptmpl( struct ldap_disptmpl *tmplist,
      struct ldap_disptmpl *tmpl );;
```

Parameters

This function has the parameters listed in Table 17-139.

Table 17-139 `ldap_next_disptmpl()` Function Parameters

Parameter	Description
<code>tmpl</code>	Defines a template from the template list <code>tmplist</code> . A NULL pointer is returned if <code>tmpl</code> is the last template in the list.
<code>tmplistp</code>	Pointer to a list of template data structures typically obtained by calling <code>ldap_init_templates()</code> .

Returns

- If successful, 0 is returned and `tmplistp` is configured.
- Upon error:
 - `LDAP_TMPL_ERR_VERSION` if `buf` points to data that is newer than can be handled.
 - `LDAP_TMPL_ERR_MEM` if there is a memory allocation problem.
 - `LDAP_TMPL_ERR_SYNTAX` if there is a problem with the format of the templates buffer or file.
 - `LDAP_TMPL_ERR_FILE` if the file cannot be read.

Description

ldap_next_disptmpl() returns the template following the previous one in the list of templates pointed to by the parameter tmplistp. tmplistp is typically obtained by calling ldap_init_templates().

See Also

ldap_first_disptmpl()

ldap_next_entry()

The ldap_next_entry() function returns a pointer to the LDAPMessage structure representing the next directory entry in a chain of search results.

Syntax

```
#include <ldap.h>
LDAPMessage * ldap_next_entry( LDAP *ld, LDAPMessage *entry );
```

Parameters

This function has the parameters listed in Table 17-140.

Table 17-140 ldap_next_entry() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
entry	Pointer to an LDAPMessage structure in a chain of search results.

Returns

One of the following values:

- If successful, returns the pointer to the next LDAPMessage structure of the type LDAP_RES_SEARCH_ENTRY in a chain of search results.
- If no more LDAPMessage structures of the type LDAP_RES_SEARCH_ENTRY are in the chain or if the function is unsuccessful, returns a NULLMSG.

Description

The `ldap_next_entry()` function returns a pointer to the `LDAPMessage` structure representing the next directory entry in a chain of search results. You can use this function in conjunction with the `ldap_first_entry()` function to iterate through the directory entries in a chain of search results. These functions skip over any messages in the chain that do not have the type `LDAP_RES_SEARCH_ENTRY` as messages containing directory entries have the type `LDAP_RES_SEARCH_ENTRY`.

For more information, see “Sorting the Search Results” on page 131 in Chapter 5, “Searching a LDAP Directory.”

See Also

`ldap_first_entry()`

ldap_next_message()

The `ldap_next_message()` function returns a pointer to the next `LDAPMessage` structure in a chain of search results.

Syntax

```
#include <ldap.h>
LDAPMessage * ldap_next_message( LDAP *ld, LDAPMessage *msg );
```

Parameters

This function has the parameters listed in Table 17-141.

Table 17-141 `ldap_next_message()` Function Parameters

Parameter	Description
<code>ld</code>	Pointer to an LDAP structure containing information about the connection to the LDAP server.
<code>msg</code>	Pointer to an <code>LDAPMessage</code> structure in a chain of search results.

Returns

One of the following values:

- If successful, returns the pointer to the next `LDAPMessage` structure in a chain of search results.
- If no more `LDAPMessage` structures are in the chain or if the function is unsuccessful, returns a `NULLMSG`.

ldap_next_reference()

Description

The `ldap_next_message()` function returns a pointer to the next `LDAPMessage` structure in a chain of search results. You can use this function in conjunction with the `ldap_first_message()` function to iterate through the chain of search results. You can also call the `ldap_msgtype()` function to determine if each message contains a matching entry (a message of the type `LDAP_RES_SEARCH_ENTRY`) or a search reference (a message of the type `LDAP_RES_SEARCH_REFERENCE`).

For more information, see “Sorting the Search Results” on page 131 in Chapter 5, “Searching a LDAP Directory.”

Example

See Code Example 5-4 on page 115 and Code Example 5-7 on page 119 of Chapter 5, “Searching a LDAP Directory.”

See Also

`ldap_first_message()`, `ldap_msgtype()`

ldap_next_reference()

The `ldap_next_reference()` function returns a pointer to the `LDAPMessage` structure representing the next search reference in a chain of search results.

Syntax

```
#include <ldap.h>
LDAPMessage * ldap_next_reference( LDAP *ld, LDAPMessage *ref );
```

Parameters

This function has the parameters listed in Table 17-142.

Table 17-142 `ldap_next_reference()` Function Parameters

Parameter	Description
<code>ld</code>	Pointer to an LDAP structure containing information about the connection to the LDAP server.
<code>msg</code>	Pointer to an <code>LDAPMessage</code> structure in a chain of search results.

Returns

One of the following values:

- If successful, returns the pointer to the next LDAPMessage structure of the type LDAP_RES_SEARCH_REFERENCE in a chain of search results.
- If no more LDAPMessage structures of the type LDAP_RES_SEARCH_REFERENCE are in the chain or if the function is unsuccessful, returns a NULLMSG.

Description

The ldap_next_reference() function returns a pointer to the LDAPMessage structure representing the next search reference in a chain of search results. You can use this function in conjunction with the ldap_first_reference() function to iterate through the search references in a chain of search results. These functions skip over any messages in the chain that do not have the type LDAP_RES_SEARCH_REFERENCE as messages containing search references have the type LDAP_RES_SEARCH_REFERENCE.

For more information, see “Sorting the Search Results” on page 131 in Chapter 5, “Searching a LDAP Directory.”

See Also

ldap_first_reference()

ldap_next_searchobj()

The ldap_next_searchobj() function returns the following search preference in a defined list.

Syntax

```
#include <ldap.h>
struct ldap_searchobj * ldap_next_searchobj
    ( struct ldap_searchobj *solist, struct ldap_searchobj *so );
```

Parameters

This function has the parameters listed in Table 17-143.

Table 17-143 ldap_next_searchobj() Function Parameters

Parameter	Description
solist	Pointer to a list of data structures, typically obtained by calling ldap_init_searchprefs().
so	Pointer to the most recent search object returned in the template list solist. The search object returned by ldap_next_searchobj() follows this one.

Returns

- If successful, a pointer to an ldap_searchobj structure.
- A NULL pointer is returned if so is the last entry in the list.

Description

The search object returned by ldap_next_searchobj() follows the one defined by the so parameter.

See Also

ldap_init_searchprefs(), ldap_first_searchobj()

ldap_next_tmplcol()

The ldap_next_tmplcol() function returns a pointer to the following item (in the column) within a template.

Syntax

```
#include <disptmpl.h>
struct ldap_tmplitem * ldap_next_tmplcol( struct ldap_disptmpl *tmpl,
    struct ldap_tmplitem *row, struct ldap_tmplitem *col );
```

Parameters

This function has the parameters listed in Table 17-144.

Table 17-144 ldap_next_tmplcol() Function Parameters

Parameter	Description
tmpl	The name of the template to be retrieved.
row	The row in which the item is to be retrieved from.
col	The column in which the item is to be retrieved from.

Returns

- If successful, a pointer to an ldap_tmplitem structure.
- A NULL pointer on error or if col was the last item.

Description

ldap_next_tmplcol() returns a pointer to the next item (in the column) of the row, defined by row, within the template defined by tmpl.

See Also

ldap_first_tmplcol()

ldap_next_tmplrow()

The `ldap_next_tmplrow()` function returns a pointer to the following row in a template.

Syntax

```
#include <disptmpl.h>
struct ldap_tmplitem * ldap_next_tmplrow( struct ldap_disptmpl *tmpl,
    struct ldap_tmplitem *row );
```

Parameters

This function has the parameters listed in Table 17-145.

Table 17-145 ldap_next_tmplrow() Function Parameters

Parameter	Description
tmpl	The name of the template to be retrieved.
row	The row in the template to be retrieved.

Returns

- If successful, a pointer to an `ldap_tmplitem` structure.
- A NULL pointer on error.

Description

`ldap_next_tmplrow()` returns a pointer to the row that follows the one defined by row in the template defined by `tmpl`.

ldap_oc2template()

The `ldap_oc2template()` function obtains a pointer to the correct `ldap_disptmpl` structure.

Syntax

```
#include <disptmpl.h>
struct ldap_disptmpl * ldap_oc2template( char **oclist,
    struct ldap_disptmpl *tmplist );
```

Parameters

This function has the parameters listed in Table 17-146.

Table 17-146 ldap_oc2template() Function Parameters

Parameter	Description
oclist	A NULL-terminated array of strings that contains the values of the objectClass attribute of the entry.
tmplistp	Pointer to a list of template data structures.

Returns

- Pointer to the first template where all of the object classes listed in one of the template's dt_oclist elements are contained in oclist.
- A NULL pointer if no appropriate template is found.

Description

ldap_oc2template() searches tmplist for the best template to use to display an entry that has a specific set of objectClass values.

ldap_open()

CAUTION This function is deprecated and should not be used. Please use ldap_init().

The ldap_open() function opens a connection to an LDAP server and allocates an LDAP structure which is used to identify the connection and maintain per-connection information.

Syntax

```
#include <ldap-deprecated.h>
ldap_open( const char *host, int port );
```

Parameters

This function has the parameters listed in Table 17-148.

Table 17-147 ldap_open() Function Parameters

Parameter	Description
host	The hostname on which the LDAP server is running. It may contain a blank-separated list of hosts to try to connect to, and each host may optionally be of the form <i>host:port</i> . If present, <i>port</i> overrides the port parameter. Upon successfully making a connection to an LDAP server, ldap_open() returns a pointer to an LDAP structure, which should be passed to subsequent calls (ldap_bind(), ldap_search(), and so forth).
port	The port number to which to connect. If the default IANA-assigned port of 389 is desired, LDAP_PORT should be specified as the value of port.

Description

Please use the newer version of this function, ldap_init().

ldap_parse_entrychange_control()

The ldap_parse_entrychange_control() function examines a list of controls returned from a persistent search operation, retrieves an entry change control, and parses that control for information (such as the type of change made to the entry and the change number).

NOTE

This function implements an extension to the LDAPv3. Entry change notification is an optional feature; it may not be supported on all LDAP servers. Call this function when interacting with LDAP servers that support this LDAPv3 extension. See “Determining Supported Extended Operations” on page 268 of Chapter 15, “Extended Operations” to determine if the server supports extended operations.

Syntax

```
#include <ldap.h>
int ldap_parse_entrychange_control( LDAP *ld,
    LDAPControl **ctrls, int *chgtypep, char **prevdn,
    int *chgnumpresentp, long *chgnump );
```

Parameters

This function has the parameters listed in Table 17-148.

Table 17-148 ldap_parse_entrychange_control() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
ctrlp	An array of controls returned by the server. You obtain these controls by calling the ldap_get_entry_controls() function on an entry returned by the server.
changetypes	Pointer to an integer specifying the type of change made to the entry. This field can have one of the following values: - LDAP_CHANGETYPE_ADD specifies that the entry was added to the directory. - LDAP_CHANGETYPE_DELETE specifies that the entry was deleted from the directory. - LDAP_CHANGETYPE_MODIFY specifies that the entry was modified. - LDAP_CHANGETYPE_MODDN specifies that the DN or RDN of the entry was changed (a modify RDN or modify DN operation was performed).
prevdn	Pointer to the previous DN of the entry, if the changetypes argument is LDAP_CHANGETYPE_MODDN. (If the changetypes argument has a different value, this argument is set to NULL.) When done, you can free this by calling the ldap_memfree() function.
chgnumpresentp	Pointer to an integer specifying whether or not the change number is included in the control. The parameter can have the following possible values: 0 specifies that the change number is not included. - A non-zero value specifies that the change number is included and is available as the chgnump argument.
chgnump	Pointer to the change number identifying the change made to the entry, if chgnumpresentp points to a non-zero value.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_DECODING_ERROR if an error occurred when BER-decoding the control.

Description

The ldap_parse_entrychange_control() function can be called:

- To parse an entry returned from a persistent search operation and retrieve an entry change control.

- After receiving an entry from a persistent search and retrieving the controls from the entry. (Call ldap_get_entry_controls() to get the controls).

See Also

ldap_create_persistentsearch_control(), ldap_get_entry_controls()

ldap_parse_extended_result()

The ldap_parse_extended_result() function parses the results of an LDAP extended operation and retrieves the object identifier (OID) and data returned by the server.

NOTE

This function implements an extension to the LDAPv3. Extended operations might not be supported on all LDAP servers. Call this function only when interacting with LDAP servers that support this LDAPv3 extension. See “Determining Supported Extended Operations” on page 268 of Chapter 15, “Extended Operations” to determine if the server supports extended operations.

Syntax

```
#include <ldap.h>
int ldap_parse_extended_result( LDAP *ld, LDAPMessage *res,
    char **retoidp, struct berval **retdatap, int freeit );
```

Parameters

This function has the parameters listed in Table 17-149.

Table 17-149 ldap_parse_extended_result() Function Parameters

Parameters	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
res	Pointer to the LDAPMessage structure containing the results of an LDAP operation.
retoidp	Pointer to the OID returned by the server after performing the extended operation. When done, you can free this by calling the ldap_memfree() function.
retdatap	Pointer to the pointer to a berval structure containing the data returned by the server after performing the extended operation. When done, you can free this by calling the ber_bvfree() function.

Table 17-149 ldap_parse_extended_result() Function Parameters (Continued)

Parameters	Description
freeit	Specifies whether or not to free the results of the operation (the LDAPMessage structure specified by the res argument). The parameter can have the following possible values: • 0 specifies that the result should not be freed. • A non-zero value specifies that the result should be freed.

Returns

One of the following values, which indicates the result of parsing the server’s response:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if any of the arguments are invalid.
- LDAP_DECODING_ERROR if an error occurred when decoding the BER-encoded results from the server.
- LDAP_NOT_SUPPORTED if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see “Specifying the LDAP Version of Your Client” on page 65 of Chapter 3, “Writing an LDAP Client.”)

NOTE This value does not apply to the operation itself.

Description

ldap_parse_extended_result() parses the server’s response to an extended operation. After you call the ldap_extended_operation() and the ldap_result() functions, you can pass the results to ldap_parse_extended_result(). This function gets the following data from the server’s response:

- The extended operation OID received from the server is passed back as the retoidp argument.
- The data received from the server is passed back in the berval structure as the retdatap argument.
- The LDAP result code for the LDAP extended operation is placed in the ld structure. You can get the result code by calling the ldap_get_lderrno() function.

For a list of possible result codes for an LDAP extended operation, see `ldap_extended_operation_s()`.

NOTE The LDAP server must support the extended operation. Sun Java System Directory Server supports a server plug-in interface that you can use to add support for extended operations. For details, see the *Sun ONE Directory Server Plug-In Programming Guide*.

See Also

`ldap_extended_operation_s()`, `ldap_get_lderrno()`

ldap_parse_reference()

The `ldap_parse_reference()` function parses search references from the results received from an LDAP server.

NOTE Search references are part of the LDAPv3. When calling this function, make sure that you are working with a server that supports the LDAPv3.

Syntax

```
#include <ldap.h>
int ldap_parse_reference( LDAP *ld, LDAPMessage *ref,
    char ***referralsp, LDAPControl ***serverctrlsp,
    int freeit );
```

Parameters

This function has the parameters listed in Table 17-150.

Table 17-150 `ldap_parse_reference()` Function Parameters

Parameter	Description
<code>ld</code>	Pointer to an LDAP structure containing information about the connection to the LDAP server.
<code>ref</code>	Pointer to an LDAPMessage structure of the type <code>LDAP_RES_SEARCH_REFERENCE</code> .
<code>referralsp</code>	Pointer to an array of strings representing the referrals found by an LDAP search operation and returned by the server (applicable only if the LDAP operation was a search operation). When done, you can free this by calling the <code>ldap_value_free()</code> function.

Table 17-150 ldap_parse_reference() Function Parameters (*Continued*)

Parameter	Description
serverctrls p	Pointer to an array of LDAPControl structures, which represent the LDAPv3 server controls returned by the server. When done, you can free this by calling the ldap_controls_free() function.
freeit	Specifies whether or not to free the results of the operation (the LDAPMessage structure specified by the res argument). The parameter can have the following possible values: • 0 specifies that the result should not be freed. • A non-zero value specifies that the result should be freed.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if any of the arguments are invalid.
- LDAP_DECODING_ERROR if an error occurred when decoding the BER-encoded results from the server.
- LDAP_NO_MEMORY if memory cannot be allocated.

Description

The ldap_parse_reference() function parses the referral URLs from an LDAPMessage structure of the type LDAP_RES_SEARCH_REFERENCE.

ldap_parse_result()

The ldap_parse_result() function parses the results of an LDAP operation received from an LDAP server.

Syntax

```
#include <ldap.h>
int ldap_parse_result( LDAP *ld, LDAPMessage *res,
    int *errcodep, char **matcheddn, char **errmsgp,
    char ***referralsp, LDAPControl ***serverctrlsp, int freeit);
```

Parameters

This function has the parameters listed in Table 17-151.

Table 17-151 ldap_parse_result() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
res	Pointer to the LDAPMessage structure containing the results of an LDAP operation.
errcodep	Pointer to the LDAP result code specifying the result of the operation.
matcheddn	Pointer to a string specifying the portion of a DN that finds an existing entry (in cases where the server cannot find the entry specified by a DN). See “Receiving the Matching Portion of a DN” on page 92 of Chapter 4, “Using the LDAP API” for details. When done, you can free this by calling the ldap_memfree() function.
errmsgp	Pointer to an additional error message string sent from the server. When done, you can free this by calling the ldap_memfree() function.
referralsp	Pointer to an array of strings representing the referrals returned by the server. When done, you can free this by calling the ldap_value_free() function.
serverctrls p	Pointer to an array of LDAPControl structures, which represent the LDAPv3 server controls returned by the server. When done, you can free this by calling the ldap_controls_free() function.
freeit	Specifies whether or not to automatically free the results of the operation (the LDAPMessage structure specified by the res argument). The parameter can have the following possible values: • 0 specifies that the result should not be freed. • A non-zero value specifies that the result should be freed.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_NO_RESULTS_RETURNED if the specified LDAPMessage structure does not contain the result of an LDAP operation (for example, if it contains an entry, search reference, or chain of search results instead of the result of an LDAP operation).
- LDAP_MORE_RESULTS_TO_RETURN if the result in the LDAPMessage structure is part of a chain of results and the last result is not included.
- LDAP_PARAM_ERROR if any of the arguments are invalid.
- LDAP_DECODING_ERROR if an error occurred when decoding the BER-encoded results from the server.

- LDAP_NO_MEMORY if memory cannot be allocated.

Description

The `ldap_parse_result()` function parses the results of an LDAP operation (received from an LDAP server) and retrieves:

- The LDAP result code that indicates the result of the LDAP operation (`errcodep`).
- An additional error message (optional) sent by the server (`errmsgp`).
- The portion of the DN that finds an entry, if the server is unable to find an entry from a DN that you specify (`matcheddn`). (See “Receiving the Matching Portion of a DN” on page 92 in Chapter 4, “Using the LDAP API” for details.)
- A set of referrals, if the server does not contain the entries that you’ve specified and if the server is configured to refer clients to other servers (`referralsp`).
- A set of server response controls that are relevant to the operation (`serverctrlsp`).

Calling this function creates an array of `LDAPControl` structures that you can pass to subsequent API functions (such as the `ldap_parse_sort_control()` function).

For more information, see “Performing an Asynchronous Authentication Operation” on page 69 in Chapter 3, “Writing an LDAP Client.”

NOTE

This function is not intended to be used to parse entries and search references. Use the `ldap_msgtype()` function to determine the type of result contained in the `LDAPMessage` structure. If the result is an entry returned as a search result, call the `ldap_first_entry()` function to retrieve the entry. If the result is a search reference, call the `ldap_parse_reference()` function to retrieve the reference.

Example

See the examples as follow:

- For an example of parsing the results of an asynchronous LDAP add operation, see Code Example 7-9 on page 157.
- For an example of parsing the results of an asynchronous LDAP modify operation, see Code Example 7-17 on page 170.
- For an example of parsing the results of an asynchronous LDAP delete operation, see Code Example 7-21 on page 177.
- For an example of parsing the results of an asynchronous LDAP modify DN operation, see Code Example 7-24 on page 183.

See the example in Chapter 3, “Writing an LDAP Client” as follows:

- For an example of parsing the results of an asynchronous LDAP bind operation, see Code Example 3-9 on page 70.

See the example in Chapter 4, “Using the LDAP API” as follows:

- For an example of parsing the results of an asynchronous LDAP bind operation, see Code Example 4-5 on page 80.

See the example in Chapter 5, “Searching a LDAP Directory” as follows:

- For an example of parsing the results of an asynchronous LDAP search operation, see Code Example 5-7 on page 119.

See Also

ldap_result()

ldap_parse_sasl_bind_result()

The ldap_parse_sasl_bind_result() function parses the results of an LDAP SASL bind operation and retrieves data returned by the server.

NOTE SASL authentication is part of the LDAPv3. When calling this function, make sure that you are working with a server that supports the LDAPv3.

Syntax

```
#include <ldap.h>
int ldap_parse_sasl_bind_result( LDAP *ld, LDAPMessage *res,
    struct berval **servercredp, int freeit );
```

Parameters

This function has the parameters listed in Table 17-152.

Table 17-152 ldap_parse_sals_bind_result() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
res	Pointer to the LDAPMessage structure containing the results of an LDAP operation.

Table 17-152 ldap_parse_sals_bind_result() Function Parameters *(Continued)*

Parameter	Description
servercredp	Pointer to a pointer to an berval structure containing any challenge or credentials returned by the server. When done, you can free this by calling the ber_bvfree() function.
freeit	Specifies whether or not to free the results of the operation (the LDAPMessage structure specified by the res argument). The parameter can have the following possible values: • 0 specifies that the result should not be freed. • A non-zero value specifies that the result should be freed.

Returns

One of the following values, which indicates the result of parsing the server’s response:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if any of the arguments are invalid.
- LDAP_DECODING_ERROR if an error occurred when decoding the BER-encoded message.
- LDAP_NOT_SUPPORTED if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see “Specifying the LDAP Version of Your Client” on page 65 of Chapter 3, “Writing an LDAP Client.”)

NOTE	This value does not apply to the SASL bind operation itself)

Description

After you call the ldap_sasl_bind() function and the ldap_result() function, you can pass the results to ldap_parse_sasl_bind_result() for parsing. This function gets the following data from the server’s response:

- The challenge or credentials sent back from the server are passed back in the berval structure as the servercredp argument.

- The LDAP result code for the SASL bind operation is placed in the `ld` structure. You can get the result code by calling the `ldap_get_lderrno()` function. (If the result code is `LDAP_SASL_BIND_IN_PROGRESS`, you can call `ldap_sasl_bind()` again to send a response to the server's challenge and call `ldap_result()` and `ldap_parse_sasl_bind_result()` again to get the next challenge from the server. For a list of possible result codes for an LDAP SASL bind operation, see `ldap_sasl_bind()`).

NOTE The LDAP server must support authentication through SASL mechanisms. Sun Java System Directory Server supports a plug-in interface that you can use to add SASL support to the server. For details, see the *Sun ONE Directory Server Plug-In Programming Guide*.

See Also

`ldap_sasl_bind()`, `ldap_get_lderrno()`

ldap_parse_sort_control()

The `parse_sort_control()` function parses the result returned from a search operation that used a server control for sorting search results.

NOTE This function implements an extension to the LDAPv3. Server-pside sorting is an optional feature; it may not be supported on all LDAP servers. Call this function when interacting with LDAP servers that support this LDAPv3 extension. See “Determining Supported Extended Operations” on page 268 of Chapter 15, “Extended Operations” to determine if the server supports extended operations.

Syntax

```
#include <ldap.h>
int ldap_parse_sort_control( LDAP *ld, LDAPControl **ctrls,
    unsigned long *result, char **attribute );
```

Parameters

This function has the parameters listed in Table 17-153.

Table 17-153 `ldap_parse_sort_control()` Function Parameters

Parameters	Description
<code>ld</code>	Pointer to an LDAP structure containing information about the connection to the LDAP server.

Table 17-153 ldap_parse_sort_control() Function Parameters (*Continued*)

Parameters	Description
ctrls	An array of controls returned by the server. You obtain these controls by calling the ldap_parse_result() function on the set of results returned by the server.
result	Pointer to the sort control result code retrieved by this function.
attribute	If the sorting operation fails, the function sets this to point to the name of the attribute that caused the failure. When done, you can free this by calling the ldap_memfree() function.

Returns

One of the following values, which indicates the result of parsing the server’s response:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if any of the arguments are invalid.
- LDAP_NO_MEMORY if memory cannot be allocated to decode the control returned by the server.
- LDAP_DECODING_ERROR if an error occurred when decoding the BER-encoded message.
- LDAP_CONTROL_NOT_FOUND if no control can be found in the response returned from the server.

Description

Call the ldap_parse_sort_control() function as part of the process of retrieving sorted search results from a server. First, though, call ldap_result() to get the results, and ldap_parse_result() to parse the server controls from the results.

See Also

ldap_create_sort_control()

ldap_parse_virtuallist_control()

The ldap_parse_virtuallist_control() function parses the result returned from a search operation that used a server control for virtual list views.

NOTE	This function implements an extension to the LDAPv3. A <i>virtual list view</i> is an optional LDAP server feature that may not be supported on all LDAP servers. For more information about this control, see “Using the Virtual List View Control” on page 245 of Chapter 13, “Working with LDAP Controls.” For information on determining if a server supports this or other LDAPv3 controls, see “Determining LDAPv3 Support” on page 215 of Chapter 10, “Retrieving Server Information.”

Syntax

```
#include <ldap.h>
int ldap_parse_virtuallist_control( LDAP *ld,
    LDAPControl **ctrls, unsigned long *target_posp,
    unsigned long *list_sizep, int *errcodep );
```

Parameters

This function has the parameters listed in Table 17-154.

Table 17-154 ldap_parse_virtuallist_control() Function Parameters

Parameters	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
ctrls	An array of controls returned by the server. You obtain these controls by calling the ldap_parse_result() function on the set of results returned by the server.
target_posp	Pointer to an unsigned long that is set by the function. The function sets this to the index or offset of the selected entry in the list of entries.
list_sizep	Pointer to an unsigned long that is set by the function. The function sets this to the number of entries in the total number of entries in the entire list (not just the subset).
errcodep	Pointer to the sort control result code retrieved by this function.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if any of the arguments are invalid.
- LDAP_NO_MEMORY if memory cannot be allocated to decode the control returned by the server.

- LDAP_DECODING_ERROR if an error occurred when decoding the BER-encoded message.
- LDAP_CONTROL_NOT_FOUND if no control can be found in the response returned from the server.
- LDAP_NOT_SUPPORTED if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see “Specifying the LDAP Version of Your Client” on page 65 of Chapter 3, “Writing an LDAP Client.”)

Description

The `ldap_parse_virtuallist_control()` function can be called:

- As part of the process of retrieving a subset of entries from a list when working with a virtual list view box.
- After, calling `ldap_result()` to get the results, and `ldap_parse_result()` to parse the server controls from the results.

See Also

`ldap_create_virtuallist_control()`

ldap_perror()

CAUTION This function is deprecated and should not be used. It is included in `ldap-deprecated.h` for backward-compatibility. Please use `ldap_get_lderrno()` instead.

The `ldap_perror()` function prints, to standard output, the last LDAP error message.

Syntax

```
#include <ldap-deprecated.h>
void ldap_perror( LDAP *ld, const char *s );
```

Parameters

This function has the parameters listed in Table 17-155.

Table 17-155 ldap_perror() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
s	Optional text to print out before printing the error message.

Description

Please use the newer version of this function, ldap_get_lderrno().

For more information, see “Printing Out Error Messages” on page 92 of Chapter 4, “Using the LDAP API.”

Example

Code Example 17-43 prints out an error message if the search operation cannot complete successfully.

Code Example 17-43 ldap_perror() Code Example

```

...
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, my_filter,
  get_attr, 0, &result ) != LDAP_SUCCESS ) {
  ldap_perror( ld, "ldap_search_s" );
  return( 1 );
...

```

See Also

ldap_get_lderrno(), ldap_err2string(), ldap_result2error(),
ldap_set_lderrno()

ldap_rename()

The ldap_rename() function changes the DN of an entry in the directory asynchronously.

NOTE ldap_rename() is a new version of the ldap_modrdn2() function. If you are writing a new LDAP client, you should call ldap_rename().

Syntax

```
#include <ldap.h>
int ldap_rename( LDAP *ld, const char *dn, const char *newrdn,
    const char *newparent, int deleteoldrdn,
    LDAPControl **serverctrls, LDAPControl **clientctrls,
    int *msgidp );
```

Parameters

This function has the parameters listed in Table 17-156.

Table 17-156 ldap_rename() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
dn	DN of the entry to rename.
newrdn	New Relative Distinguished Name (RDN) to assign to the entry.
newparent	DN of the new parent entry you want to move the entry under. Pass NULL if you do not want to move the entry to a different location in the directory tree.
deleteoldrdn	Specifies whether or not the old RDN is retained as an attribute of the entry. For example, an entry has the following values for the cn attribute: cn: Barbara Jensen cn: Babs Jensen If you change the RDN to cn=Barbie Jensen" and pass 1 as deleteoldrdn, the resulting entry has the following values: cn: Barbie Jensen cn: Babs Jensen If instead you pass 0 as deleteoldrdn, the Barbara Jensen value is not removed from the entry: cn: Barbie Jensen cn: Babs Jensen cn: Barbara Jensen So, if this is a non-zero value, the old RDN is not retained as a value in the entry. If 0, the old RDN is retained as an attribute in the entry.
serverctrls	Pointer to an array of LDAPControl structures representing LDAP server controls that apply to this LDAP operation. If you do not want to pass any server controls, specify NULL for this argument.
clientctrls	Pointer to an array of LDAPControl structures representing LDAP client controls that apply to this LDAP operation. If you do not want to pass any client controls, specify NULL for this argument.

Table 17-156 ldap_rename() Function Parameters (*Continued*)

Parameter	Description
msgidp	Pointer to an integer that will be set to the message ID of the LDAP operation. To check the result of this operation, call the ldap_result() and ldap_parse_result() functions.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if any of the arguments are invalid.
- LDAP_NO_MEMORY if memory cannot be allocated to decode the control returned by the server.
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the request.
- LDAP_SERVER_DOWN if the LDAP server did not receive the request or if the connection to the server was lost.
- LDAP_NOT_SUPPORTED if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see “Specifying the LDAP Version of Your Client” on page 65 of Chapter 3, “Writing an LDAP Client.”)

Description

The ldap_rename() changes the DN of an entry in the directory and allows you to move the entry under a different parent entry in the directory tree. ldap_rename() is an asynchronous function; it does not directly return results. In order to get the results of the LDAP rename operation, you need to call the ldap_result() function and the ldap_parse_result() function. If you want the results to be returned directly by the function, call the synchronous function ldap_rename_s() instead.

For more information on changing the DN of an entry, see “Changing the DN of an Entry” on page 181 of Chapter 17, “Function Reference.” For a list of possible result codes for an LDAP rename operation, see the RETURNS section of the ldap_rename_s() function.

Example

See Code Example 7-24 on page 183 in Chapter 7, “Adding, Modifying and Deleting Entries.”

See Also

ldap_rename_s(), ldap_result(), ldap_parse_result()

ldap_rename_s()

The ldap_rename_s() function changes the DN of an entry in the directory synchronously.

NOTE ldap_rename_s() is a new version of the ldap_modrdn2_s() function. If you are writing a new LDAP client, you should call ldap_rename_s().

Syntax

```
#include <ldap.h>
int ldap_rename_s( LDAP *ld, const char *dn, const char *newrdn,
    const char *newparent, int deleteoldrdn,
    LDAPControl **serverctrls, LDAPControl **clientctrls );
```

Parameters

This function has the parameters listed in Table 17-157.

Table 17-157 ldap_rename() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
dn	DN of the entry to rename.
newrdn	New Relative Distinguished Name (RDN) to assign to the entry.
newparent	DN of the new parent entry you want to move the entry under. Pass NULL if you do not want to move the entry to a different location in the directory tree.

Table 17-157 ldap_rename() Function Parameters (*Continued*)

Parameter	Description
deleteoldrdn	Specifies whether or not the old RDN is retained as an attribute of the entry. For example, an entry has the following values for the cn attribute: <pre>cn: Barbara Jensen cn: Babs Jensen</pre> If you change the RDN to cn=Barbie Jensen and pass 1 as deleteoldrdn, the resulting entry has the following values: <pre>cn: Barbie Jensen cn: Babs Jensen</pre> If instead you pass 0 as deleteoldrdn, the Barbara Jensen value is not removed from the entry: <pre>cn: Barbie Jensen cn: Babs Jensen cn: Barbara Jensen</pre> So, if this is a non-zero value, the old RDN is not retained as a value in the entry. If 0, the old RDN is retained as an attribute in the entry.
serverctrls	Pointer to an array of LDAPControl structures representing LDAP server controls that apply to this operation. If you do not want to pass any server controls, specify NULL for this argument.
clientctrls	Pointer to an array of LDAPControl structures representing LDAP client controls that apply to this operation. If you do not want to pass any client controls, specify NULL for this argument.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if any of the arguments are invalid.
- LDAP_NO_MEMORY if memory cannot be allocated to decode the control returned by the server.
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the request.
- LDAP_SERVER_DOWN if the LDAP server did not receive the request or if the connection to the server was lost.
- LDAP_LOCAL_ERROR if an error occurred when receiving the results from the server.

- `LDAP_DECODING_ERROR` if an error occurred when decoding the BER-encoded results from the server.
- `LDAP_NOT_SUPPORTED` if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see “Specifying the LDAP Version of Your Client” on page 65 of Chapter 3, “Writing an LDAP Client.”)

NOTE

Sun Java System Directory Server and other LDAP server products may send result codes in addition to those described. For example, the server may have loaded a plug-in that returns custom result codes. Check your LDAP server documentation for other result codes.

Description

The `ldap_rename_s()` changes the DN of an entry in the directory and allows you to move the entry under a different parent entry in the directory tree. The function `ldap_rename_s()` is synchronous; it directly returns the results of the operation. If you want to perform other operations while waiting for the results of this operation, call the asynchronous function `ldap_rename()`.

For more information on changing the DN of an entry, see “Changing the DN of an Entry” on page 181 of Chapter 17, “Function Reference.”

Example

See Code Example 7-25 on page 185 in Chapter 7, “Adding, Modifying and Deleting Entries.”

See Also

`ldap_rename()`

ldap_result()

The function `ldap_result()` waits for and returns the result of an LDAP operation initiated by one of the asynchronous LDAP API functions.

Syntax

```
#include <ldap.h>
int ldap_result( LDAP *ld, int msgid, int all,
                struct timeval *timeout, LDAPMessage **result );
```


Parameters

This function has the parameters listed in Table 17-158.

Table 17-158 ldap_result() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
msgid	Asynchronous functions return a unique message ID. This parameter takes message ID of the operation for which you want the results. To check any operation, pass LDAP_RES_ANY as the value of this parameter.
all	Specifies how the results of a search are returned. This parameter can have the following values: 0 specifies that the results are returned one entry at a time, using separate calls to ldap_result(). - A non-zero value specifies that all results are returned at the same time (after the final search result is obtained by the library).
timeout	Specifies a maximum interval to wait for the selection to complete. If timeout is a NULL pointer, the select blocks indefinitely. To effect a poll, the timeout parameter should be a non-NULL pointer, pointing to a zero-valued timeval structure.
result	Result of the operation. To interpret the results, pass this to the LDAP parsing routines, such as ldap_result2error(), ldap_parse_result(), and ldap_first_entry().

Returns

One of the following values:

- LDAP_RES_BIND indicates that the LDAPMessage structure contains the result of an LDAP bind operation.
- LDAP_RES_SEARCH_ENTRY indicates that the LDAPMessage structure contains an entry found during an LDAP search operation.
- LDAP_RES_SEARCH_REFERENCE indicates that the LDAPMessage structure contains an LDAPv3 search reference (a referral to another LDAP server) found during an LDAP search operation.
- LDAP_RES_SEARCH_RESULT indicates that the LDAPMessage structure contains the result of an LDAP search operation.
- LDAP_RES_MODIFY indicates that the LDAPMessage structure contains the result of an LDAP modify operation.

- LDAP_RES_ADD indicates that the LDAPMessage structure contains the result of an LDAP add operation.
- LDAP_RES_DELETE indicates that the LDAPMessage structure contains the result of an LDAP delete operation.
- LDAP_RES_MODDN or LDAP_RES_RENAME indicates that the LDAPMessage structure contains the result of an LDAP modify DN operation.
- LDAP_RES_COMPARE indicates that the LDAPMessage structure contains the result of an LDAP compare operation.
- LDAP_RES_EXTENDED indicates that the LDAPMessage structure contains the result of an LDAPv3 extended operation.
- -1 indicates that an error occurred. The error code is set in the LDAP structure. To get the error code, call the ldap_get_lderrno() function. (See Chapter 18, “Result Codes” for a complete listing of error codes.)
- 0 indicates that the operation has timed out.

Description

For more information, see “Performing an Asynchronous Authentication Operation” on page 69 in Chapter 3, “Writing an LDAP Client” and “To Retrieve the Server Response” on page 78 in Chapter 4, “Using the LDAP API.”

Example

See the examples in the following sections:

- For examples of an asynchronous LDAP add operation, see “Asynchronous Add Operation” on page 156 of Chapter 7, “Adding, Modifying and Deleting Entries.”
- For examples of getting the results of an asynchronous LDAP modify operation, see “Asynchronous Modify Operation” on page 169 of Chapter 7, “Adding, Modifying and Deleting Entries.”
- For examples of getting the results of an asynchronous LDAP delete operation, see “Asynchronous Delete Operation” on page 176 of Chapter 7, “Adding, Modifying and Deleting Entries.”
- For examples of getting the results of an asynchronous LDAP modify DN operation, see “Changing the DN of an Entry” on page 181 of Chapter 7, “Adding, Modifying and Deleting Entries.”
- For examples of getting the results of an asynchronous LDAP search operation, see “Getting Results Asynchronously” on page 118 of Chapter 5, “Searching a LDAP Directory.”

- For examples of getting the results of an asynchronous LDAP bind operation, see “Performing an Asynchronous Authentication Operation” on page 69 of Chapter 3, “Writing an LDAP Client.”

See Also

ldap_add_ext(), ldap_compare_ext(), ldap_delete_ext(), ldap_modify_ext(), ldap_rename(), ldap_simple_bind(), ldap_url_search()

ldap_result2error()

CAUTION This function is deprecated and should not be used. It is included in ldap-deprecated.h for backward-compatibility. Please use the newer version, ldap_parse_result().

The ldap_result2error() function returns the corresponding error code for a result produced by the ldap_result() and ldap_search_s() functions.

Syntax

```
#include <ldap-deprecated.h>
int ldap_result2error( LDAP *ld, LDAPMessage *r, int freeit );
```

Parameters

This function has the parameters listed in Table 17-159.

Table 17-159 ldap_result2error() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
r	Pointer to the LDAPMessage structure representing the results returned by the ldap_result() or ldap_search() function.
freeit	Specifies whether or not the result should be freed after the error code is extracted. The parameter can have the following possible values: • 0 specifies that the result should not be freed. • A non-zero value specifies that the result should be freed.

Returns

One of the following values:

- If successful, sets the error code and other error information in the LDAP structure and returns the error code. See Chapter 18, “Result Codes” for a complete listing of error codes.
- If unsuccessful, returns LDAP_PARAM_ERROR.

Example

See ldap_result().

See Also

ldap_parse_result(), ldap_get_lderrno(), ldap_err2string(), ldap_result(), ldap_set_lderrno()

ldap_sasl_bind()

The ldap_sasl_bind() function authenticates your client to an LDAP server using an Simple Authentication and Security Layer (SASL) mechanism.

NOTE

The LDAP server must support authentication through SASL. Sun Java System Directory Server supports a server plug-in interface that you can use to add SASL support to the server. For details, see the *Sun ONE Directory Server Plug-In Programming Guide* (<http://docs.sun.com/doc/816-6702-10>).

Syntax

```
#include <ldap.h>
int ldap_sasl_bind( LDAP *ld, const char *dn,
    const char *mechanism, const struct berval *cred,
    LDAPControl **serverctrls, LDAPControl **clientctrls,
    int *msgidp );
```

Parameters

This function has the parameters listed in Table 17-160.

Table 17-160 ldap_sasl_bind() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
dn	DN of the user who wants to authenticate. For anonymous authentication, set this to NULL.
mechanism	Name of the SASL mechanism that you want to use for authentication.

Table 17-160 ldap_sasl_bind() Function Parameters (*Continued*)

Parameter	Description
cred	Pointer to the berval structure containing the credentials that you want to use for authentication.
serverctrls	Pointer to an array of LDAPControl structures representing LDAP server controls that apply to this operation. If you do not want to pass any server controls, specify NULL for this argument.
clientctrls	Pointer to an array of LDAPControl structures representing LDAP client controls that apply to this operation. If you do not want to pass any client controls, specify NULL for this argument.
msgidp	Pointer to an integer that will be set to the message ID of the LDAP operation. To check the result of this operation, call ldap_result() and ldap_parse_sasl_bind_result() functions.

Returns

One of the following values:

- LDAP_SUCCESS if the SASL bind request was sent successfully.
- LDAP_PARAM_ERROR if an invalid parameter was passed to the function.
- LDAP_NOT_SUPPORTED if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see “Specifying the LDAP Version of Your Client” on page 65 of Chapter 3, “Writing an LDAP Client.”)
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the request to send to the server.
- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_SERVER_DOWN if the LDAP server did not receive the request or if the connection to the server was lost.

Description

The ldap_sasl_bind() function authenticates your client to an LDAP server by using a specified SASL mechanism. ldap_sasl_bind() is an asynchronous function; it does not directly return results. If you want the results to be returned directly by the function, call the synchronous function ldap_sasl_bind_s(). In order to get the results of the LDAP SASL bind operation, call the ldap_result(), the ldap_parse_sasl_bind_result(), and the ldap_get_lderrno() functions.

For more details, see “Binding and Authenticating to an LDAP Server” on page 66 of Chapter 3, “Writing an LDAP Client.” For additional information on authenticating through SASL mechanisms, see Chapter 12, “Using SASL Authentication.” For a list of possible result codes for an LDAP SASL bind operation, see `ldap_sasl_bind_s()`.

See Also

`ldap_result()`, `ldap_parse_sasl_bind_result()`, `ldap_get_lderrno()`, `ldap_sasl_bind_s()`

ldap_sasl_bind_s()

The `ldap_sasl_bind_s()` function authenticates your client to an LDAP server synchronously using an SASL mechanism.

NOTE

The LDAP server must support authentication through SASL. Sun Java System Directory Server supports a server plug-in interface that you can use to add SASL support to the server. For details, see the *Sun ONE Directory Server Plug-In Programming Guide* (<http://docs.sun.com/doc/816-6702-10>).

Syntax

```
#include <ldap.h>
int ldap_sasl_bind_s( LDAP *ld, const char *dn,
    const char *mechanism, const struct berval *cred,
    LDAPControl **serverctrls, LDAPControl **clientctrls,
    struct berval **servercredp );
```

Parameters

This function has the parameters listed in Table 17-161.

Table 17-161 `ldap_sasl_bind_s()` Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
dn	DN of the user who wants to authenticate. For anonymous authentication, set this to NULL.
mechanism	Name of the SASL mechanism that you want to use for authentication.
cred	Pointer to the <code>berval</code> structure containing the credentials that you want to use for authentication.

Table 17-161 ldap_sasl_bind_s() Function Parameters (*Continued*)

Parameter	Description
serverctrls	Pointer to an array of LDAPControl structures representing LDAP server controls that apply to this operation. If you do not want to pass any server controls, specify NULL for this argument.
clientctrls	Pointer to an array of LDAPControl structures representing LDAP client controls that apply to this operation. If you do not want to pass any client controls, specify NULL for this argument.
servercredp	Pointer to a pointer to a berval structure containing any credentials returned by the server. When done, you can free this by calling the ber_alloc() function.

Returns

One of the following values:

- LDAP_SUCCESS if the SASL bind request was sent successfully.
- LDAP_PARAM_ERROR if an invalid parameter was passed to the function.
- LDAP_NOT_SUPPORTED if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see “Specifying the LDAP Version of Your Client” on page 65 of Chapter 3, “Writing an LDAP Client.”)
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the request to send to the server.
- LDAP_DECODING_ERROR if an error occurred when the LDAP API library was decoding the BER-encoded results received from the server.
- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_SERVER_DOWN if the LDAP server did not receive the request or if the connection to the server was lost.
- LDAP_LOCAL_ERROR if an error occurred when receiving the results from the server.

NOTE

Sun Java System Directory Server and other LDAP server products may send result codes in addition to those described. For example, the server may have loaded a plug-in that returns custom result codes. Check your LDAP server documentation for other result codes.

Description

The `ldap_sasl_bind_s()` function authenticates your client to an LDAP server by using a specified SASL mechanism. It is a synchronous function, which directly returns the results of the operation. If you want to perform other operations while waiting for the results of this operation, call the asynchronous function `ldap_sasl_bind()`. After authenticating a client through SASL, an LDAP server can return a set of credentials in the results. The `servercredp` argument points to this value.

For more details, see “Binding and Authenticating to an LDAP Server” on page 66 of Chapter 3, “Writing an LDAP Client.” For additional information on authenticating through SASL mechanisms, see Chapter 12, “Using SASL Authentication.”

See Also

`ldap_sasl_bind()`

ldap_search()

The `ldap_search()` function searches the directory asynchronously.

NOTE This is an older function that is included in the SDK for backward-compatibility. If you are writing a new LDAP client, use `ldap_search_ext()` instead.

Syntax

```
#include <ldap.h>
int ldap_search( LDAP *ld, const char *base, int scope,
                const char* filter, char **attrs, int attrsonly );
```

Parameters

This function has the parameters detailed in Table 17-162.

Table 17-162 `ldap_search()` Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
base	DN of the entry that serves as the starting point for the search. For example, setting base to <code>dc=example,dc=com</code> restricts the search to entries at <code>example.com</code> .

Table 17-162 ldap_search() Function Parameters (*Continued*)

Parameter	Description
scope	Scope of the search, which can be one of the following values: - LDAP_SCOPE_BASE searches the entry specified by base. - LDAP_SCOPE_ONELEVEL searches all entries one level beneath the entry specified by base. - LDAP_SCOPE_SUBTREE searches the entry specified by base and all entries at all levels beneath the entry specified by base.
filter	String representation of the filter to apply in the search. You can specify simple filters with the following syntax: (attributetype=attributevalue) For details on the syntax for filters, see “Defining a Search Filter” on page 109 of Chapter 5, “Searching a LDAP Directory.”
attrs	A NULL-terminated array of attribute types to return from entries that match filter. If you specify a NULL, all attributes will be returned.
attrsonly	Specifies whether or not attribute values are returned along with the attribute types. This parameter can have the following values: 0 specifies that both attribute types and attribute values are returned. 1 specifies that only attribute types are returned.

Returns

Returns the message ID of the ldap_search() operation.

NOTE

To check the result of this operation, call ldap_result() and ldap_result2error(). See ldap_search_ext_s() for a list of possible result codes.

Description

Please use the newer version of this function, ldap_search_ext().

Example

Code Example 17-44 searches a directory.

Code Example 17-44 ldap_search() Code Example

```
#include "examples.h"

static void do_other_work();
unsigned long global_counter = 0;
```

Code Example 17-44 ldap_search() Code Example (*Continued*)

```

int
main( int argc, char **argv )
{
    LDAP *ld;
    LDAPMessage *result, *e;
    BerElement *ber;
    char *a, *dn;
    char **vals;
    int i, rc, finished, msgid;
    int num_entries = 0;
    struct timeval zerotime;

    zerotime.tv_sec = zerotime.tv_usec = 0L;

    /* get a handle to an LDAP connection */
    if ( (ld = ldap_init( MY_HOST, MY_PORT )) == NULL ) {
        perror( "ldap_init" );
        return( 1 );
    }
    /* authenticate to the directory as nobody */
    if ( ldap_simple_bind_s( ld, NULL, NULL ) != LDAP_SUCCESS ) {
        ldap_perror( ld, "ldap_simple_bind_s" );
        return( 1 );
    }
    /* search for all entries with surname of Jensen */
    if ( (msgid = ldap_search( ld, MY_SEARCHBASE, LDAP_SCOPE_SUBTREE,
        MY_FILTER, NULL, 0 )) == -1 ) {
        ldap_perror( ld, "ldap_search" );
        return( 1 );
    }

    /* Loop, polling for results until finished */
    finished = 0;
    while ( !finished ) {
        /*
         * Poll for results. We call ldap_result with the "all" parameter
         * set to zero. This causes ldap_result() to return exactly one
         * entry if at least one entry is available. This allows us to
         * display the entries as they are received.
         */
        result = NULL;
        rc = ldap_result( ld, msgid, 0, &zerotime, &result );
        switch ( rc ) {
            case -1:
                /* some error occurred */
                ldap_perror( ld, "ldap_result" );
                return( 1 );
            case 0:
                /* Timeout was exceeded. No entries are ready for retrieval. */
                if ( result != NULL ) {
                    ldap_msgfree( result );
                }
                break;
            default:

```

Code Example 17-44 ldap_search() Code Example (*Continued*)

```

/*
 * Either an entry is ready for retrieval, or all entries have
 * been retrieved.
 */
if (( e = ldap_first_entry( ld, result )) == NULL ) {
    /* All done */
    finished = 1;
    if ( result != NULL ) {
        ldap_msgfree( result );
    }
    continue;
}
/* for each entry print out name + all attrs and values */
num_entries++;
if (( dn = ldap_get_dn( ld, e )) != NULL ) {
    printf( "dn: %s\n", dn );
    ldap_memfree( dn );
}
for ( a = ldap_first_attribute( ld, e, &ber );
      a != NULL; a = ldap_next_attribute( ld, e, ber ) ) {
    if (( vals = ldap_get_values( ld, e, a )) != NULL ) {
        for ( i = 0; vals[ i ] != NULL; i++ ) {
            printf( "%s: %s\n", a, vals[ i ] );
        }
        ldap_value_free( vals );
    }
    ldap_memfree( a );
}
if ( ber != NULL ) {
    ldap_ber_free( ber, 0 );
}
printf( "\n" );
ldap_msgfree( result );
}
/* Do other work here while you are waiting... */
do_other_work();
}

/* All done. Print a summary. */
printf( "%d entries retrieved. I counted to %ld "
        "while waiting.\n", num_entries, global_counter );
ldap_unbind( ld );
return( 0 );
}

/*
 * Perform other work while polling for results. */
static void
do_other_work()
{
    global_counter++;
}

```

See Also
ldap_search_ext()

ldap_search_ext()

The ldap_search_ext() function searches the directory asynchronously.

NOTE ldap_search_ext() is a new version of the ldap_search() function. If you are writing a new LDAP client, you should call ldap_search_ext().

Syntax

```
#include <ldap.h>
int ldap_search_ext( LDAP *ld, const char *base, int scope,
    const char *filter, char **attrs, int attrsonly,
    LDAPControl **serverctrls, LDAPControl **clientctrls,
    struct timeval *timeoutp, int sizelimit, int *msgidp );
```

Parameters
This function has the parameters listed in Table 17-163.

Table 17-163 ldap_search_ext() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
base	DN of the entry that serves as the starting point for the search. For example, setting base to dc=example,dc=com restricts the search to entries at example.com.
scope	Scope of the search, which can be one of the following values: - LDAP_SCOPE_BASE searches the entry specified by base. - LDAP_SCOPE_ONELEVEL searches all entries one level beneath the entry specified by base. - LDAP_SCOPE_SUBTREE searches the entry specified by base and all entries at all levels beneath the entry specified by base.
filter	String representation of the filter to apply in the search. You can specify simple filters with the following syntax: (attributetype=attributevalue) For details on the syntax for filters, see “Defining a Search Filter” on page 109 of Chapter 5, “Searching a LDAP Directory.”

Table 17-163 ldap_search_ext() Function Parameters (*Continued*)

Parameter	Description
attrs	A NULL-terminated array of attribute types to return from entries that match filter. If you specify a NULL, all attributes will be returned.
attrsonly	Specifies whether or not attribute values are returned along with the attribute types. This parameter can have the following values: 0 specifies that both attribute types and attribute values are returned. 1 specifies that only attribute types are returned.
serverctrls	Pointer to an array of LDAPControl structures representing LDAP server controls that apply to this operation. If you do not want to pass any server controls, specify NULL for this argument.
clientctrls	Pointer to an array of LDAPControl structures representing LDAP client controls that apply to this operation. If you do not want to pass any client controls, specify NULL for this argument.
timeoutp	Pointer to a timeval structure specifying the maximum time to wait for the results of the search. Pass NULL to use the default time limit for the current connection. To specify an infinite time limit, set the tv_sec and tv_usec fields in the timeval structure to 0. See “Setting Search Preferences” on page 112 of Chapter 5, “Searching a LDAP Directory.”
sizelimit	Maximum number of results to return in the search. Pass -1 to use the default size limit for the current connection.
msgidp	Pointer to an integer that will be set to the message ID of the LDAP operation.

Returns

One of the following values:

- LDAP_SUCCESS if the SASL bind request was sent successfully.
- LDAP_PARAM_ERROR if an invalid parameter was passed to the function.
- LDAP_NOT_SUPPORTED if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see “Specifying the LDAP Version of Your Client” on page 65 of Chapter 3, “Writing an LDAP Client.”)
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the request to send to the server.
- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_SERVER_DOWN if the LDAP server did not receive the request or if the connection to the server was lost.

Description

The `ldap_search_ext()` function searches the directory for matching entries. `ldap_search_ext()` is an asynchronous function; it does not directly return results. If you want the results to be returned directly by the function, call the synchronous function `ldap_search_ext_s()`. You can also use this function to pass LDAP server controls to the server if you want the server to sort the results or if you want to request a persistent search. (See `ldap_create_sort_control()` and `ldap_create_persistentsearch_control()` for more information.) In order to get the results of the `ldap_search_ext()` operation, you need to call the `ldap_result()` and the `ldap_parse_result()` functions.

For more information on searching the directory, see Chapter 5, “Searching a LDAP Directory.” For a list of possible result codes for an LDAP search operation, see `ldap_search_ext_s()`.

Example

See Code Example 5-7 on page 119 of Chapter 5, “Searching a LDAP Directory.”

See Also

`ldap_search_ext_s()`, `ldap_result()`, `ldap_parse_result()`

ldap_search_ext_s()

The `ldap_search_ext_s()` function searches the directory synchronously.

NOTE `ldap_search_ext_s()` is a new version of the `ldap_search_s()` function. If you are writing a new LDAP client, you should call `ldap_search_ext_s()`.

Syntax

```
#include <ldap.h>
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 *timeoutp, int sizelimit, LDAPMessage **res );
```

Parameters

This function has the parameters listed in Table 17-164.

Table 17-164 ldap_search_ext_s() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
base	DN of the entry that serves as the starting point for the search. For example, setting base to dc=example,dc=com restricts the search to entries at example.com.
scope	Scope of the search, which can be one of the following values: - LDAP_SCOPE_BASE searches the entry specified by base. - LDAP_SCOPE_ONELEVEL searches all entries one level beneath the entry specified by base. - LDAP_SCOPE_SUBTREE searches all entries at all levels beneath the entry specified by base.
filter	String representation of the filter to apply in the search. You can specify simple filters with the following syntax: <i>(attributetype=attributevalue)</i> For details on the syntax for filters, see “Defining a Search Filter” on page 109 of Chapter 5, “Searching a LDAP Directory.”
attrs	A NULL-terminated array of attribute types to return from entries that match filter. If you specify a NULL, all attributes will be returned.
attrsonly	Specifies whether or not attribute values are returned along with the attribute types. This parameter can have the following values: 0 specifies that both attribute types and attribute values are returned. 1 specifies that only attribute types are returned.
serverctrls	Pointer to an array of LDAPControl structures representing LDAP server controls that apply to this operation. If you do not want to pass any server controls, specify NULL for this argument.
clientctrls	Pointer to an array of LDAPControl structures representing LDAP client controls that apply to this operation. If you do not want to pass any client controls, specify NULL for this argument.
timeoutp	Pointer to a timeval structure specifying the maximum time to wait for the results of the search. See “Setting Search Preferences” on page 112 of Chapter 5, “Searching a LDAP Directory.”
sizelimit	Maximum number of results to return in the search.
res	Results of the search (when the call is completed).

Returns

One of the following values:

- `LDAP_SUCCESS` if the SASL bind request was sent successfully.
- `LDAP_PARAM_ERROR` if an invalid parameter was passed to the function.
- `LDAP_NOT_SUPPORTED` if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see “Specifying the LDAP Version of Your Client” on page 65 of Chapter 3, “Writing an LDAP Client.”)
- `LDAP_FILTER_ERROR` if an error occurred when parsing and BER-encoding the search filter specified by the filter argument.
- `LDAP_TIMEOUT` if the search exceeded the time specified by the timeoutp argument.
- `LDAP_ENCODING_ERROR` if an error occurred when BER-encoding the request to send to the server.
- `LDAP_DECODING_ERROR` if an error occurred when the LDAP API library was decoding the BER-encoded results received from the server.
- `LDAP_NO_MEMORY` if memory cannot be allocated.
- `LDAP_SERVER_DOWN` if the LDAP server did not receive the request or if the connection to the server was lost.
- `LDAP_LOCAL_ERROR` if an error occurred when receiving the results from the server.

NOTE Sun Java System Directory Server and other LDAP server products may send result codes in addition to those described. For example, the server may have loaded a plug-in that returns custom result codes. Check your LDAP server documentation for other result codes.

Description

The `ldap_search_ext_s()` searches the directory for matching entries. The function `ldap_search_ext_s()` is synchronous; it directly returns the results of the operation. If you want to perform other operations while waiting for the results of this operation, call the asynchronous function `ldap_search_ext()`. You can also use `ldap_search_ext_s()` to pass LDAP server controls to the server if you want the server to sort the results or if you want to request a persistent search. (See `ldap_create_sort_control()` and `ldap_create_persistentsearch_control()` for more information.)

For more information on searching the directory, see Chapter 5, “Searching a LDAP Directory.”

Example

See Code Example 5-4 on page 115 of Chapter 5, “Searching a LDAP Directory.”

See Also

ldap_search_ext()

ldap_search_s()

The `ldap_search_s()` function searches the directory synchronously.

NOTE	This is an older function that is included in the SDK for backward-compatibility. If you are writing a new LDAP client, use <code>ldap_search_ext_s()</code> instead.

Syntax

```
#include <ldap.h>
int ldap_search_s( LDAP *ld, const char *base, int scope,
    const char* filter, char **attrs, int attrsonly,
    LDAPMessage **res );
```

Parameters

This function has the parameters listed in Table 17-165.

Table 17-165 ldap_search_s() Function Parameters

Parameter	Description
<code>ld</code>	Pointer to an LDAP structure containing information about the connection to the LDAP server.
<code>base</code>	DN of the entry that serves as the starting point for the search. For example, setting <code>base</code> to <code>dc=example,dc=com</code> restricts the search to entries at <code>example.com</code> .
<code>scope</code>	Scope of the search, which can be one of the following values: <code>LDAP_SCOPE_BASE</code> searches the entry specified by <code>base</code>. <code>LDAP_SCOPE_ONELEVEL</code> searches all entries one level beneath the entry specified by <code>base</code>. <code>LDAP_SCOPE_SUBTREE</code> searches all entries at all levels beneath the entry specified by <code>base</code>.

Table 17-165 ldap_search_s() Function Parameters (*Continued*)

Parameter	Description
filter	String representation of the filter to apply in the search. You can specify simple filters with the following syntax: (attributetype=attributevalue) For details on the syntax for filters, see “Defining a Search Filter” on page 109 of Chapter 5, “Searching a LDAP Directory.”
attrs	A NULL-terminated array of attribute types to return from entries that match filter. If you specify a NULL, all attributes will be returned.
attrsonly	Specifies whether or not attribute values are returned along with the attribute types. This parameter can have the following values: • 0 specifies that both attribute types and attribute values are returned. • 1 specifies that only attribute types are returned.
res	Results of the search (when the call is completed).

Returns

For a list of possible result codes for an LDAP search operation, see ldap_search_ext_s().

Description

Please use the newer version of this function, ldap_search_ext_s().

Example

Code Example 17-45 searches the directory for all people whose surname (last name) is “Jensen”.

Code Example 17-45 ldap_search_s() Code Example

```
#include "examples.h"

int main( int argc, char **argv )
{
    LDAP *ld;
    LDAPMessage *result, *e;
    BerElement *ber;
    char *a, *dn;
    char **vals;
    int i;
    /* get a handle to an LDAP connection */
    if ( (ld = ldap_init( MY_HOST, MY_PORT )) == NULL ) {
        perror( "ldap_init" );
        return( 1 );
    }
}
```

Code Example 17-45 ldap_search_s() Code Example (Continued)

```

/* authenticate to the directory as nobody */
if ( ldap_simple_bind_s( ld, NULL, NULL ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_simple_bind_s" );
    return( 1 );
}
/* search for all entries with surname of Jensen */
if ( ldap_search_s( ld, MY_SEARCHBASE, LDAP_SCOPE_SUBTREE,
    MY_FILTER, NULL, 0, &result ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_search_s" );
    return( 1 );
}
/* for each entry print out name + all attrs and values */
for ( e = ldap_first_entry( ld, result ); e != NULL;
    e = ldap_next_entry( ld, e ) ) {
    if ( (dn = ldap_get_dn( ld, e )) != NULL ) {
        printf( "dn: %s\n", dn );
        ldap_memfree( dn );
    }
    for ( a = ldap_first_attribute( ld, e, &ber );
        a != NULL; a = ldap_next_attribute( ld, e, ber ) ) {
        if ((vals = ldap_get_values( ld, e, a )) != NULL ) {
            for ( i = 0; vals[i] != NULL; i++ ) {
                printf( "%s: %s\n", a, vals[i] );
            }
            ldap_value_free( vals );
        }
        ldap_memfree( a );
    }
    if ( ber != NULL ) {
        ldap_ber_free( ber, 0 );
    }
    printf( "\n" );
}
ldap_msgfree( result );
ldap_unbind( ld );
return( 0 );
}

```

See Also

ldap_search_ext_s()

ldap_search_st()

The `ldap_search_st()` function searches the directory synchronously within a specified time limit.

NOTE This is an older function that is included in the SDK for backward-compatibility. If you are writing a new LDAP client, use `ldap_search_ext_s()` instead.

Syntax

```
#include <ldap.h>
int ldap_search_st( LDAP *ld, const char *base, int scope,
    const char* filter, char **attrs, int attrsonly,
    struct timeval *timeout, LDAPMessage **res );
```

Parameters

This function has the parameters listed in Table 17-166.

Table 17-166 ldap_search_st() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
base	DN of the entry that serves as the starting point for the search. For example, setting base to dc=example,dc=com restricts the search to entries at example.com.
scope	Scope of the search, which can be one of the following values: - LDAP_SCOPE_BASE searches the entry specified by base. - LDAP_SCOPE_ONELEVEL searches all entries one level beneath the entry specified by base. - LDAP_SCOPE_SUBTREE searches all entries at all levels beneath the entry specified by base.
filter	String representation of the filter to apply in the search. You can specify simple filters with the following syntax: (attributetype=attributevalue) For details on the syntax for filters, see “Defining a Search Filter” on page 109 of Chapter 5, “Searching a LDAP Directory.”
attrs	A NULL-terminated array of attribute types to return from entries that match filter. If you specify a NULL, all attributes will be returned.
attrsonly	Specifies whether or not attribute values are returned along with the attribute types. This parameter can have the following values: 0 specifies that both attribute types and attribute values are returned. 1 specifies that only attribute types are returned.
timeout	Maximum time to wait for the results of the search.
res	Results of the search (when the call is completed).

Returns

For a list of possible result codes for an LDAP search operation, see `ldap_search_ext_s()`.

Description

Please use the newer version of this function, `ldap_search_ext_s()`.

See Also

`ldap_search_ext_s()`

ldap_set_filter_additions()

The `ldap_set_filter_additions()` function sets a prefix to be prepended and a suffix to be appended to all filters returned by the `ldap_getfirstfilter()` and `ldap_getnextfilter()` function calls.

Syntax

```
#include <ldap.h>
int ldap_set_filter_additions( LDAPFiltDesc *lfdp, char *prefix,
                             char *suffix );
```

Parameters

This function has the parameters listed in Table 17-167.

Table 17-167 ldap_set_filter_additions() Function Parameters

Parameter	Description
<code>lfdp</code>	Pointer to an <code>LDAPFiltDesc</code> structure.
<code>prefix</code>	Prefix to prepend to all filters. If <code>NULL</code> , no prefix is prepended.
<code>suffix</code>	Suffix to append to all filters. If <code>NULL</code> , no suffix is appended.

Returns

One of the following values:

- `LDAP_SUCCESS` if successful.
- If unsuccessful, returns an LDAP error code. See Chapter 18, “Result Codes” for a complete listing.

Example

Code Example 17-46 loads the filter configuration file named `myfilters.conf` into memory and adds the prefix "`(&(objectClass=person))`" and the suffix "`)`" to each filter retrieved.

Code Example 17-46 ldap_set_filter_additions() Code Example

```
#include <ldap.h>
...
LDAPFiltDesc *lfdp;
char *filter_file = "myfilters.conf";
char *prefix = "&(objectClass=person)";
char *suffix = ")";
int rc;
...
lfdp = ldap_init_getfilter( filter_file );
rc = ldap_set_filter_additions( lfdp, prefix, suffix );
if ( rc != LDAP_SUCCESS ) {
    printf( "Error setting filter prefix and suffix\n");
    return( rc );
}
...
```

See Also

ldap_getfirstfilter(), ldap_getnextfilter()

ldap_setfilteraffixes()

CAUTION This function is deprecated and should not be used. It is included in `ldap-deprecated.h` for backward-compatibility. Please use `ldap_set_filter_additions()`.

For more information, see the man page for `ldap_setfilteraffixes`.

See Also

ldap_set_filter_additions()

ldap_set_lderrno()

The `ldap_set_lderrno()` function sets an error code and information about an error in an LDAP structure.

Syntax

```
#include <ldap.h>
int ldap_set_lderrno( LDAP *ld, int e, char *m, char *s );
```

Parameters

This function has the parameters listed in Table 17-168.

Table 17-168 `ldap_set_lderrno()` Function Parameters

Parameter	Description
<code>ld</code>	Pointer to an LDAP structure containing information about the connection to the LDAP server.
<code>e</code>	The error code that you want to set.
<code>m</code>	In the event that an entry for a specified DN cannot be found, set this parameter to the portion of the DN that identifies an existing entry. (See “Receiving the Matching Portion of a DN” on page 92 of Chapter 4, “Using the LDAP API” for details.)
<code>s</code>	The text of the error message that you want associated with this error code.

Returns

One of the following values:

- `LDAP_SUCCESS` if successful.
- If unsuccessful, returns an LDAP error code. See Chapter 18, “Result Codes” for a complete listing.

Description

The `ldap_set_lderrno()` function sets an error code and information about an error in an LDAP structure. You can call this function to set error information that will be retrieved by subsequent `ldap_get_lderrno()` function calls.

For more information, see “Setting Error Codes” on page 87 of Chapter 4, “Using the LDAP API.”

Example

Code Example 17-47 attempts to perform an operation. If the operation fails, the `LDAP_PARAM_ERROR` error code is placed in the LDAP structure.

Code Example 17-47 ldap_set_lderrno() Code Example

```
#include <ldap.h>
int rc;
char *errmsg = "Invalid parameter";
...
if ( ldap_my_function() != LDAP_SUCCESS ) {
    rc = ldap_set_lderrno( ld, LDAP_PARAM_ERROR, NULL, errmsg );
    if ( rc != LDAP_SUCCESS ) {
        printf( "Error: %d\nError code could not be set.\n", rc );
    }
    return( rc );
}
...
```

See Also

ldap_err2string(), ldap_perror(), ldap_result2error()

ldap_set_option()

The function ldap_set_option() sets session preferences in the LDAP structure.

Syntax

```
#include <ldap.h>
int ldap_set_option( LDAP *ld, int option, const void *optdata );
```

Parameters

This function has the parameters listed in Table 17-169.

Table 17-169 ldap_set_option() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server. If NULL, you are setting the default options that will apply to any new LDAP connection handles that are subsequently created.
option	Option that you want to set. See Table 17-170 for available options of this parameter.
optdata	Pointer to the value of the option that you want to set. Available data types are also listed in Table 17-170.

The option parameter can have one of the values listed in Table 17-170.

Table 17-170 Options for ldap_set_option()

Option	Description
LDAP_OPT_API_FEATURE_INFO	Retrieves information about the revision of a supported LDAP feature. This option is READ-ONLY and cannot be set. The data type for the optdata parameter is (LDAPAPIFeatureInfo *).
LDAP_OPT_API_INFO	Retrieves information about the API implementation at execution time (API version, protocol version, the names of supported API extensions with their vendor name version, etc.). For details on the structure returned, refer to the ldap.h header file. This option is READ-ONLY and cannot be set. The data type for the optdata parameter is (LDAPAPIInfo *).
LDAP_OPT_CLIENT_CONTROLS	Pointer to an array of LDAPControl structures representing the LDAPv3 client controls you want sent with every request by default. The data type for the optdata parameter is (LDAPControl ***).
LDAP_OPT_DESC	Socket descriptor underlying the main LDAP connection. The LBER_SOCKET data type depends on the platform that you are using: • int in UNIX. • SOCKET in Windows. The data type for the optdata parameter is (LBER_SOCKET *).
LDAP_OPT_DEREF	Determines how aliases work during a search. optdata can be one of the following values: • LDAP_DEREF_NEVER specifies that aliases are never dereferenced. • LDAP_DEREF_SEARCHING specifies that aliases are dereferenced when searching under the base object (but not when finding the base object). • LDAP_DEREF_FINDING specifies that aliases are dereferenced when finding the base object (but not when searching under the base object). • LDAP_DEREF_ALWAYS specifies that aliases are always dereferenced when finding and searching under the base object. The data type for the optdata parameter is (int *).
LDAP_OPT_DNS_FN_PTRS	DEPRECATED OPTION: Lets you use alternate DNS functions for getting the host entry of the LDAP server. The data type for the optdata parameter is (struct ldap_dns_fns *).

Table 17-170 Options for ldap_set_option() (*Continued*)

Option	Description
LDAP_OPT_ERROR_NUMBER	Retrieves the result code for the most recent LDAP error that occurred in this session. The data type for the optdata parameter is (int *).
LDAP_OPT_ERROR_STRING	Retrieves the error message returned with the result code for the most recent LDAP error that occurred in this session. The data type for the optdata parameter is (char **).
LDAP_OPT_EXTRA_THREAD_FN_PTRS	Lets you specify the locking and semaphore functions that you want called when getting results from the server. (See Chapter 14, "Writing Multithreaded Clients" for details.) The data type for the optdata parameter is (struct ldap_extra_thread_fns *).
LDAP_OPT_HOST_NAME	Sets the host name (or list of hosts) for the primary LDAP server. The data type for the optdata parameter is (char **).
LDAP_OPT_IO_FN_PTRS	DEPRECATED OPTION: Lets you use alternate communication stacks. The data type for the optdata parameter is (struct ldap_io_fns *).
LDAP_OPT_MATCHED_DN	Gets the matched DN value returned with the most recent LDAP error that occurred for this session. The data type for the optdata parameter is (char **).
LDAP_OPT_MEMALLOC_FN_PTRS	Gets a pointer to the callback structure which you previously set. The data type for the optdata parameter is (struct ldap_memalloc_fns ldap_io_fns *).
LDAP_OPT_PROTOCOL_VERSION	Version of the protocol supported by your client. You can specify either LDAP_VERSION2 or LDAP_VERSION3. If no version is set, the default is LDAP_VERSION2. In order to use LDAPv3 features, you need to set the protocol version to LDAP_VERSION3. The data type for the optdata parameter is (int *).
LDAP_OPT_REBIND_ARG	Lets you set the last argument passed to the routine specified by LDAP_OPT_REBIND_FN. You can also set this option by calling the ldap_set_rebind_proc() function. The data type for the optdata parameter is (void *).
LDAP_OPT_REBIND_FN	Lets you set the routine to be called when you need to authenticate a connection with another LDAP server (for example, during the course of following a referral). You can also set this option by calling the ldap_set_rebind_proc() function. The data type for the optdata parameter is (LDAP_REBINDPROC_CALLBACK *).

Table 17-170 Options for ldap_set_option() (Continued)

Option	Description
LDAP_OPT_RECONNECT	If the connection to the server is lost, determines whether or not the same connection handle should be used to reconnect to the server. By default, this option is off. For details, see “Handling Failover” on page 100 of Chapter 4, “Using the LDAP API.” optdata can be one of the following values: - LDAP_OPT_ON specifies that the same connection handle can be used to reconnect to the server. - LDAP_OPT_OFF specifies that you want to create a new connection handle to connect to the server. The data type for the optdata parameter is (int *).
LDAP_OPT_REFERRALS	Determines whether or not the client should follow referrals. By default, the client follows referrals. optdata can be one of the following values: - LDAP_OPT_ON specifies that the server should follow referrals. - LDAP_OPT_OFF specifies that the server should not follow referrals. The data type for the optdata parameter is (int *).
LDAP_OPT_REFERRAL_HOP_LIMIT	Maximum number of referrals the client should follow in a sequence. In other words, the client can only be referred this number of times before it gives up. By default, the maximum number of referrals that the client can follow in a sequence is 5 for the initial connection. This limit does not apply to individual requests that generate multiple referrals in parallel. The data type for the optdata parameter is (int *).
LDAP_OPT_RESTART	Determines whether or not LDAP I/O operations should be restarted automatically if they are prematurely aborted. optdata can be one of the following values: - LDAP_OPT_ON specifies that I/O operations should be restarted automatically. - LDAP_OPT_OFF specifies that I/O operations should not be restarted automatically. The data type for the optdata parameter is (int *).
LDAP_OPT_SERVER_CONTROLS	Pointer to an array of LDAPControl structures representing the LDAPv3 server controls you want sent with every request by default. Typically, since controls are specific to the type of request, you may want to pass the controls using operation-specific functions (such as ldap_add_ext()) instead. The data type for the optdata parameter is (LDAPControl ***).

Table 17-170 Options for ldap_set_option() (*Continued*)

Option	Description
LDAP_OPT_SIZELIMIT	Maximum number of entries that should be returned by the server in search results. The LDAP server may impose a smaller size limit than the limit you specify as the server administrator also has the ability to set this limit. The data type for the optdata parameter is (int *).
LDAP_OPT_SSL	Determines whether or not SSL is enabled. optdata can be one of the following values: - LDAP_OPT_ON specifies that SSL is enabled. - LDAP_OPT_OFF specifies that SSL is disabled. The data type for the optdata parameter is (int *).
LDAP_OPT_THREAD_FN_PTRS	Lets you specify the thread function pointers. (See Chapter 14, “Writing Multithreaded Clients” for details.) The data type for the optdata parameter is (struct ldap_thread_fns *).
LDAP_OPT_TIMELIMIT	Maximum number of seconds that should be spent by the server when answering a search request. The LDAP server may impose a shorter time limit than the limit you specify as the server administrator also has the ability to set this limit. The data type for the optdata parameter is (int *).
LDAP_X_OPT_EXTIO_FN_PTRS	Extended I/O function callback option.
LDAP_X_OPT_CONNECT_TIMEOUT	Value of a timeout (expressed in milliseconds) for nonblocking connect call. The data type for the optdata parameter is (int *).
LDAP_X_OPT_SOCKETBUF	Socket buffer structure associated to the LDAP connection.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- 1 if unsuccessful.

Description

For more information, see “Setting Session Preferences” on page 64 of Chapter 3, “Writing an LDAP Client.”

Example

Code Example 17-48 sets the maximum number of entries returned in a search.

Code Example 17-48 ldap_set_option() Code Example

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
int max_ret = 100, max_tim = 30;
char *host = "ldap.sun.com";
...
/* Initialize a session with the LDAP server ldap.sun.com:389 */
if ( ( ld = ldap_init( host, LDAP_PORT ) ) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
}

/* Set the maximum number of entries returned */
if (ldap_set_option( ld, LDAP_OPT_SIZELIMIT, &max_ret ) != LDAP_SUCCESS) {
    ldap_perror( ld, "ldap_set_option" );
    return( 1 );
}
...
```

See Also

ldap_init(), ldap_get_option()

ldap_set_rebind_proc()

The `ldap_set_rebind_proc()` function sets a *rebind*, which is called by your client to obtain authentication credentials when following a referral.

Syntax

```
#include <ldap.h>
void ldap_set_rebind_proc( LDAP *ld,
    LDAP_REBINDPROC_CALLBACK *rebindproc, void *arg );
```

Parameters

This function has the parameters listed in Table 17-171.

Table 17-171 ldap_set_rebind_proc() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
rebindproc	Pointer to a function called back to obtain bind credentials when a new server is contacted during an LDAP referral.
arg	Pointer to an additional argument that you want to pass to the rebind function.

Description

Call ldap_set_rebind_proc() to specify the rebind function. This rebind function is called by the LDAP client when following a referral to a new LDAP server. It is responsible for obtaining the credentials used to authenticate to the new server. For example, suppose LDAP server A sends a referral to your client. The referral points your client to LDAP server B. When automatically following the referral, your client calls the rebind function to obtain a DN and credentials; your client uses these to authenticate to server B. By default, if you do not call ldap_set_rebind_proc() or if you pass NULL for the rebindproc argument, your client authenticates anonymously when following referrals.

The rebind function that you specify with ldap_set_rebind_proc() should have the following prototype:

```
int LDAP_CALL LDAP_CALLBACK rebindproc( LDAP *ld, char **dn,  
    char **passwdp, int *authmethodp, int freeit, void *arg );
```

NOTE	LDAP_CALL and LDAP_CALLBACK are used to set up calling conventions, such as Pascal calling conventions on Windows. These are defined in the lber.h header file.

The following procedure explains what the rebind function is expected to do. LDAP clients that are built with Sun Java System LDAP SDK for C use this procedure when following referrals.

1. The LDAP server sends a referral back to the client.
2. The client calls the rebind function, passing 0 as the freeit argument.
3. The rebind function sets the dn, passwdp, and authmethodp arguments to point to the following information:

- The `dn` argument is set to point to the DN to be used to authenticate to the new LDAP server.
 - The `passwdp` argument is set to point to the credentials for this DN.
 - The `authmethodp` argument is set to point to the method of authentication used (for example, `LDAP_AUTH_SIMPLE`).
4. If successful, the `rebind` function returns `LDAP_SUCCESS`, and referral processing continues. (If any other value is returned, referral processing stops, and that value is returned as the result code for the original LDAP request.)
 5. The client gets the DN, credentials, and authentication method from the arguments of the `rebind` function and uses this information to authenticate to the new LDAP server.
 6. The client calls the `rebind` function again, passing `1` as the `freeit` argument.
 7. The `rebind` function frees any memory allocated earlier to specify the DN and credentials.

You need to write a `rebind` function that does the following:

- If `freeit` is `0`, set the following pointers:
 - Set `dn` to point to the DN to be used for authentication.
 - Set `passwdp` to point to the credentials to be used for authentication.
 - Set `authmethodp` to point to the method of authentication used (for example, `LDAP_AUTH_SIMPLE`).

You can also make use of the `arg` argument, which is a pointer to the argument specified in the `ldap_set_rebind_proc()` function. If successful, returns `LDAP_SUCCESS`. Otherwise, returns the appropriate LDAP error code.

- If `freeit` is `1`, free any memory that you allocated to create the DN and credentials.

After you have defined this function, pass the function name to `ldap_set_rebind_proc()` to register your `rebind` function.

NOTE In order to use the `rebind` function, the `LDAP_OPT_REFERRALS` option must be set to `LDAP_OPT_ON`, so that your client automatically follows referrals. This option is set to `LDAP_OPT_ON` by default.

Example

Code Example 17-49 demonstrates how to write and register a `rebind` function.

Code Example 17-49 ldap_set_rebind_proc() Code Example

```

#include "ldap.h"
...
/* Declare your rebind function */
int rebindproc( LDAP *ld, char **dnp, char **passwdp, int *authmethodp, int
freeit, void *arg );
...
int main( int argc, char **argv )
{
    LDAP *ld;
    /* Additional argument to be passed to the rebind function */
    char *testarg = "cn=Directory Manager";
    /* Get a handle to an LDAP connection */
    if ( (ld = ldap_init( "directory.myhost.com", 389 )) == NULL ) {
        perror( "ldap_init" );
        return( 1 );
    }
    /* Specify the function used for reauthentication on referrals */
    ldap_set_rebind_proc( ld, rebindproc, (void *)testarg );
    /* Authenticate */
    if ( ldap_simple_bind_s( ld, "uid=bjensen,ou=People,dc=example,dc=com",
        "hifalutin" ) != LDAP_SUCCESS ) {
        ldap_perror( ld, "ldap_simple_bind_s" );
        return( 1 );
    }
    ...
    /* Your code to interact with the LDAP server */
    ...
}
...
/* rebindproc is the rebind function responsible for providing the DN,
credentials, and authentication method used for authenticating the
client to other Directory Servers.
The function should set the following arguments:
- dnp should point to the DN that will be used for authentication.
- passwdp should point to the credentials used for authentication.
- authmethodp should point to the method of authentication to be used
(for example, LDAP_AUTH_SIMPLE).
The function should return LDAP_SUCCESS if successful or an LDAP
error code if an error occurs.
In order to demonstrate how the freeit argument works, this example
uses strdup() to copy the DN and password. You can also just copy
string pointers if the DN and password are already available as
global variables.
*/
int LDAP_CALL LDAP_CALLBACK rebindproc( LDAP *ld, char **dnp, char
**passwdp, int *authmethodp, int freeit, void *arg )
{
    printf( "Rebind function called.\n" );
    switch ( freeit ) {
        /* Your client calls the rebind function with freeit==1 when it needs
to free any memory you've allocated. */
        case 1:
            printf( "\tFreeing memory.\n" );

```


Code Example 17-49 ldap_set_rebind_proc() Code Example (*Continued*)

```

        if ( dnp && *dnp ) {
            free( *dnp );
        }
        if ( passwd && *passwd ) {
            free( *passwd );
        }
        break;
/* Your client calls the rebind function with freeit==0 when it needs
to get the DN, credentials, and authentication method. */
case 0:
    printf( "\tGetting DN and credentials.\n" );
    *dnp = strdup( "uid=username,o=OtherServerSuffix" );
    *passwd = strdup( "23skidoo" );
    authmethodp = LDAP_AUTH_SIMPLE;
    break;
default:
    printf( "\tUnknown value of freeit argument: %d\n", freeit );
    break;
}
/* If you successfully set the DN and credentials, you should return
LDAP_SUCCESS. (Any other return code will stop the client from
automatically following the referral. */
return LDAP_SUCCESS;
}

```

See Also

ldap_simple_bind(), ldap_simple_bind_s()

ldap_simple_bind()

The `ldap_simple_bind()` function synchronously authenticates your client to the LDAP server using a DN and a password.

Syntax

```
#include <ldap.h>
int ldap_simple_bind(LDAP *ld, const char *who, const char *passwd);
```

Parameters

This function has the parameters listed in Table 17-172.

Table 17-172 ldap_simple_bind() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
who	DN of the user who wants to authenticate. For anonymous authentication, set this or the passwd argument to NULL.
passwd	Password of the user who wants to authenticate. For anonymous authentication, set this or the who argument to NULL.

Returns

Returns the message ID of the ldap_simple_bind() operation. To check the result of this operation, call ldap_result() and ldap_result2error().

Description

The ldap_simple_bind() function authenticates to the LDAP server. It verifies that the password supplied for authentication matches the userPassword attribute of the given entry. ldap_simple_bind() is an asynchronous function; it does not directly return results. If you want the results to be returned directly by the function, call the synchronous function ldap_simple_bind_s().

NOTE	If you specify a DN but no password, your client will bind to the server anonymously. If you want a NULL password to be rejected as an incorrect password, you need to write code to perform the check before you call the ldap_simple_bind() function.

For additional information on authenticating to the LDAP server, see “Binding and Authenticating to an LDAP Server” on page 66 of Chapter 3, “Writing an LDAP Client.”

CAUTION	The use of this function may be a security threat if used on a non-secured connection as the password is transmitted in cleartext. For password-based authentication, use a secure connection (ldapssl_init() or ldapssl_tls_start_s()) or SASL Digest-MD5 mechanism (ldap_sasl_bind() or ldap_sasl_bind_s()).

Example

Code Example 17-50 calls ldap_simple_bind() to authenticate the user *Barbara Jensen* to the directory.

Code Example 17-50 ldap_simple_bind() Code Example

```

#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
char *host = "ldap.sun.com";
char *dn = "uid=bjensen,ou=People,dc=example,dc=com";
char *pw = "hifalutin";
struct timeval zerotime;
zerotime.tv_sec = zerotime.tv_usec = 0L;
...
/* Initialize a session with the LDAP server ldap.sun.com:389 */
if ( ( ld = ldap_init( host, LDAP_PORT ) ) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
}
/* Attempt to bind with the LDAP server */
msgid = ldap_simple_bind( ld, dn, pw );

/* Initialize the value returned by ldap_result() */
rc = 0;

/* While the operation is still running, do this: */
while ( rc == 0 ) {
    ... /* do other work while waiting */...

    /* Check the status of the LDAP operation */
    rc = ldap_result( ld, msgid, NULL, &zerotime, &result );
    switch( rc ) {
        /* If -1 was returned, an error occurred */
        case -1:
            ldap_perror( ld, "Error in results: " );
            return( 1 );
        /* If 0 was returned, the operation is still in progress */
        case 0:
            continue;
        /* If any other value is returned, assume we are done */
        default:
            /* Check if the "bind" operation was successful */
            if ( ldap_result2error( result ) != LDAP_SUCCESS ) {
                ldap_perror( ld, "Error binding to server: " );
                return( 1 );
            }
    }
}
...

```

See Also

ldap_simple_bind_s()

ldap_simple_bind_s()

The ldap_simple_bind() function synchronously authenticates your client to the LDAP server using a DN and a password.

Syntax

```
#include <ldap.h>
int ldap_simple_bind_s( LDAP *ld, const char *who, const char *passwd );
```

Parameters

This function has the parameters listed in Table 17-173.

Table 17-173 ldap_simple_bind_s() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
who	DN of the user who wants to authenticate. For anonymous authentication, set this or the passwd argument to NULL.
passwd	Password of the user who wants to authenticate. For anonymous authentication, set this or the who argument to NULL.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if an invalid parameter was passed to the function.
- LDAP_NOT_SUPPORTED if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see “Specifying the LDAP Version of Your Client” on page 65 of Chapter 3, “Writing an LDAP Client.”)
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the request to send to the server.
- LDAP_DECODING_ERROR if an error occurred when the LDAP API library was decoding the BER-encoded results received from the server.
- LDAP_NO_MEMORY if memory cannot be allocated.

- `LDAP_SERVER_DOWN` if the LDAP server did not receive the request or if the connection to the server was lost.
- `LDAP_LOCAL_ERROR` if an error occurred when receiving the results from the server.

NOTE Sun Java System Directory Server and other LDAP server products may send result codes in addition to those described. For example, the server may have loaded a plug-in that returns custom result codes. Check your LDAP server documentation for other result codes.

Description

The `ldap_simple_bind_s()` function authenticates to the LDAP server. It verifies that the password supplied for authentication matches the `userPassword` attribute of the given entry. `ldap_simple_bind_s()` is a synchronous function, which directly returns the results of the operation. If you want to perform other operations while waiting for the results of this operation, call the asynchronous function `ldap_simple_bind()` instead.

NOTE If you specify a DN but no password, your client will bind to the server anonymously. If you want a NULL password to be rejected as an incorrect password, you need to write code to perform the check before you call the `ldap_simple_bind_s()` function.

For additional information on authenticating to the LDAP server, see “Binding and Authenticating to an LDAP Server” on page 66 of Chapter 3, “Writing an LDAP Client.”

CAUTION The use of this function may be a security threat if used on a non-secured connection as the password is transmitted in cleartext. For password-based authentication, use a secure connection (`ldapsl_init()` or `ldapssl_tls_start_s()`) or SASL Digest-MD5 mechanism (`ldap_sasl_bind()` or `ldap_sasl_bind_s()`).

Example

Code Example 17-51 uses the synchronous `ldap_simple_bind_s()` function to authenticate to the directory as the user *Barbara Jensen*.

Code Example 17-51 ldap_simple_bind_s() Code Example

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
char *host = "ldap.sun.com";
char *dn = "uid=bjensen,ou=People,dc=example,dc=com";
char *pw = "hifalutin";
...
/* Initialize a session with the LDAP server ldap.sun.com:389 */
if ( ( ld = ldap_init( host, LDAP_PORT ) ) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
}
/* Attempt to bind with the LDAP server */
if ( ldap_simple_bind_s( ld, dn, pw ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "Authentication failed: " );
    return( 1 );
}
...
```

See Also
ldap_simple_bind()

ldap_sort_entries()

The ldap_sort_entries() function sorts a chain of entries retrieved from an LDAP search call.

Syntax
#include <ldap.h>
int ldap_sort_entries(LDAP *ld, LDAPMessage *chain, char *attr,
LDAP_CMP_CALLBACK *cmp);

Parameters
This function has the parameters listed in Table 17-174.

Table 17-174 ldap_sort_entieres() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.

Table 17-174 ldap_sort_entries() Function Parameters (*Continued*)

Parameter	Description
chain	Chain of entries returned by the ldap_result() or ldap_search_s() function.
attr	Attribute to use when sorting the results. To sort by DN instead of by attribute, use NULL.
cmp	Comparison function used when sorting the values. For details, see “Sorting Entries by an Attribute” on page 132 of Chapter 5, “Searching a LDAP Directory.”

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- If unsuccessful, returns a NULL and sets the appropriate error code in the LDAP structure. To get the error code, call ldap_get_lderrno(). (See Chapter 18, “Result Codes” for a complete listing of error codes.)

Description

The ldap_sort_entries() function sorts a chain of entries retrieved from an LDAP search call (ldap_search_s() or ldap_result()) either by DN or a specified attribute in the entries.

For additional information, see “Sorting the Search Results” on page 131 of Chapter 5, “Searching a LDAP Directory.”

Example

Code Example 17-52 sorts entries by the roomNumber attribute.

Code Example 17-52 ldap_sort_entries() Code Example

```
#include <stdio.h>
#include <string.h>
#include <ldap.h>
LDAP *ld;
LDAPMessage *result;
char *my_searchbase = "dc=example,dc=com";
char *my_filter = "(sn=Jensen)";
char *sortby = "roomNumber";
...
/* Search the directory */
if ( ldap_search_s( ld, my_searchbase, LDAP_SCOPE_SUBTREE, my_filter, NULL,
0, &result ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_search_s" );
    return( 1 );
}
```

Code Example 17-52 ldap_sort_entries() Code Example (*Continued*)

```
/* Sort the results by room number, using strcasecmp */
if ( ldap_sort_entries( ld, &result, sortby, strcasecmp ) != LDAP_SUCCESS )
{
    ldap_perror( ld, "ldap_sort_entries" );
    return( 1 );
}
...
```

See Also

ldap_multisort_entries(), ldap_result(), ldap_search_s(),
LDAP_CMP_CALLBACK

ldap_sort_strcasecmp()

The ldap_sort_strcasecmp() routine compares two strings and ignores any differences in case when comparing uppercase and lowercase characters.

Syntax

```
#include <ldap.h>
int ldap_sort_strcasecmp( const char **a, const char **b );
```

Parameters

This function has the parameters listed in Table 17-175.

Table 17-175 ldap_sort_strcasecmp() Function Parameters

Parameter	Description
a	Pointer to first string to compare
b	Pointer to second string to compare

Returns

One of the following values:

- If a is greater than b, returns a value greater than 0.
- If a is equal to b, returns 0.
- If a is less than b, returns a value less than 0.

Description

The `ldap_sort_strcasecmp()` routine compares two strings and ignores any differences in case when comparing uppercase and lowercase characters. This function is similar to the C function `strcasecmp()`. When sorting attribute values with `ldap_sort_strcasecmp()`, call `ldap_sort_strcasecmp()` to compare the attribute values.

NOTE

This function works with ASCII values only. For UTF-8 data, the comparison result is unspecified.

See Also

`ldap_sort_strcasecmp()`, `LDAP_VALCMP_CALLBACK`

ldap_sort_values()

The `ldap_sort_values()` function sorts an array of values retrieved from an `ldap_get_values()` call.

Syntax

```
#include <ldap.h>
int ldap_sort_values( LDAP *ld, char **vals,
    LDAP_VALCMP_CALLBACK cmp );
```

Parameters

This function has the parameters listed in Table 17-176.

Table 17-176 `ldap_sort_values()` Function Parameters

Parameter	Description
<code>ld</code>	Pointer to an LDAP structure containing information about the connection to the LDAP server.
<code>vals</code>	The array of values to sort.
<code>cmp</code>	Comparison function used when sorting the values. In the <code>ldap_sort_strcasecmp()</code> function, the comparison function must pass (char **) parameters. Because of this, you need to use the <code>ldap_sort_strcasecmp()</code> function, rather than a function like <code>strcasecmp()</code> .

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- If unsuccessful, returns an LDAP error code.

See Chapter 18, “Result Codes” for a complete listing of error codes.

Description

For additional information, see “Sorting the Search Results” on page 131 in Chapter 5, “Searching a LDAP Directory.”

Example

Code Example 17-53 sorts the values of attributes before printing them.

Code Example 17-53 ldap_sort_values() Code Example

```
#include <stdio.h>
#include <string.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result, *e;
BerElement *ber;
char *a, *dn;
char **vals;
int i;
char *my_searchbase = "dc=example,dc=com";
char *my_filter = "(sn=Jensen)";
...
    if ( ( vals = ldap_get_values( ld, e, a ) ) != NULL ) {
        /* Sort the values of the attribute */
        if ( ldap_sort_values( ld, vals, strcasecmp ) != LDAP_SUCCESS ) {
            ldap_perror( ld, "ldap_sort_values" );
            return( 1 );
        }
        /* Print the values of the attribute */
        for ( i = 0; vals[i] != NULL; i++ ) {
            printf( "%s: %s\n", a, vals[i] );
        }
        /* Free the values from memory */
        ldap_value_free( vals );
    }
...

```

See Also

ldap_get_values(), ldap_sort_strcasecmp(), LDAP_VALCMP_CALLBACK.

ldapssl_advclientauth_init()

The `ldapssl_advclientauth_init()` function initialize the secure parts (Security and SSL) of the runtime for use by a client application that may want to do SSL client authentication.

Syntax

```
#include <ldap_ssl.h>
int LDAP_CALL ldapssl_advclientauth_init( char *certdbpath,
    void *certdbhandle, int needkeydb, char *keydbpath,
    void *keydbhandle, int needsecmoddb, char *secmodpath,
    const int sslstrength);
```

Parameters

This function has the parameters listed in Table 17-177.

NOTE	Please see the description of the <code>sslstrength</code> value and note the potential problems that can be caused by passing in wrong host and portname values.

Table 17-177 `ldapssl_advclientauth_init()` Function Parameters

Parameter	Description
<code>certdbpath</code>	Specifies the path to the database containing certificates for your client. You can include the database filename in the path (for example, <code>/usr/mozilla/cert_file_name.db</code>).
<code>certdbhandle</code>	Pass a NULL value for this. (This parameter is not currently used.)
<code>needkeydb</code>	Specifies whether or not the private key database needs to be opened for use. This parameter can have one of the following values: - If it is a non-zero value, the function opens the private key database, which is identified by the <code>keydbpath</code> argument. - If 0, the function does not open the private key database.
<code>keydbpath</code>	Specifies the path to the database containing the private key certified by your certificate. You can include the database filename in the path (for example, <code>/usr/mozilla/key_file_name.db</code>).
<code>needsecmoddb</code>	Specifies whether or not the security module database file needs to be opened for use. This parameter can have one of the following values: - If it is a non-zero value, the function opens the security module database, which is identified by the <code>keydbpath</code> argument. - If 0, the function does not open the security modules database.

Table 17-177 ldapssl_advcliauth_init() Function Parameters (Continued)

Parameter	Description
secmodpath	Path to the database containing security modules. You can include the database filename in the path (for example, /usr/mozilla/secmod_file_name.db).
sslstrength	Specifies how the server certificate is evaluated. It takes one of the following: • LDAPSSL_AUTH_WEAK indicates that you accept the server’s certificate without checking the for certificate authority (CA) that issued the certificate. • LDAPSSL_AUTH_CERT indicates that you accept the server’s certificate only if you trust the CA that issued the certificate. • LDAPSSL_AUTH_CNCHECK indicates that you accept the server’s certificate only if you trust the CA that issued the certificate and if the value of the cn attribute is the DNS hostname of the server. If this option is selected, please ensure that the defhost parameter passed to ldapssl_init() consists of only one hostname and not a list of hosts. Furthermore, the port number must be passed via the defport parameter, and cannot be passed via a host:port option.

Returns

One of the following values:

- 0 if successful.
- -1 if unsuccessful.

Description

You can call ldapssl_advcliauth_init() to initialize your client application for SSL and certificate-based client authentication. This function is similar to ldapssl_cliauth_init() and allows you to:

- Specify the name and path of a security module database.
- Specify the method used to verify the server’s certificate.

NOTE

You must call ldapssl_advcliauth_init() before calling ldapssl_init() to connect to the server. For details, see Chapter 11, “Connecting Over SSL.”

Example

Code Example 17-54 initializes a client before connecting with a secure LDAP server.

Code Example 17-54 ldapssl_advclientauth_init() Code Example

```

#include <ldap.h>
#include <ldap_ssl.h>
#include <stdio.h>
...
/* Initialize client, using mozilla's certificate database */
if ( ldapssl_advclientauth_init( "/u/mozilla/.netscape/cert_file_name.db",
    NULL, 1, "/u/mozilla/.netscape/key_file_name.db", NULL, 1,
    "/u/mozilla/.netscape/secmod_file_name.db", LDAPSSL_AUTH_CNCHECK) < 0 ) {
    perror( "ldap_advclientauth_init" );
    return( 1 );
}
}
...

```

See Also

ldap_init(), ldapssl_clientauth_init(), ldapssl_init(),
 ldapssl_install_routines()

ldapssl_client_init()

The `ldapssl_client_init()` function initializes the secure parts (Security and SSL) of the runtime for use by your client application to connect to a secure LDAP server over SSL.

Syntax

```

#include <ldap_ssl.h>
int ldapssl_client_init( const char *certdbpath, void *certdbhandle );

```

Parameters

This function has the parameters listed in Table 17-178.

Table 17-178 ldapssl_client_init() Function Parameters

Parameter	Description
certdbpath	Specifies the path to the database containing certificates for your client. You can include the database filename in the path (for example, <code>/usr/mozilla/cert_file_name.db</code>).
certdbhandle	Pass a NULL value for this. (This parameter is not used currently.)

Returns

One of the following values:

- 0 if successful.
- -1 if unsuccessful.

Description

You can call the `ldapssl_client_init()` function to initialize your client application for SSL. It is only called once and returns 0 if all goes well. If you plan to use certificate-based authentication, you should call either the `ldapssl_clientauth_init()` or the `ldapssl_advclientauth_init()` function.

NOTE

You must call `ldapssl_client_init()` before calling the `ldapssl_init()` function to connect to the server. For details, see Chapter 11, "Connecting Over SSL."

Example

Code Example 17-55 initializes a client before connecting with a secure LDAP server.

Code Example 17-55 `ldapssl_client_init()` Code Example

```
#include <ldap.h>
#include <ldap_ssl.h>
#include <stdio.h>
...
/* Initialize client, using mozilla's certificate database */
if ( ldapssl_client_init( "/u/mozilla/.netscape/cert_file_name.db", NULL ) < 0 )
{
    printf( "Failed to initialize SSL client...\n" );
    return( 1 );
}
...
```

See Also

`ldap_init()`, `ldapssl_init()`, `ldapssl_install_routines()`

ldapssl_clientauth_init()

The `ldapssl_clientauth_init()` function initializes your client application to connect to a secure LDAP server over SSL and to use certificate-based client authentication.

Syntax

```
#include <ldap_ssl.h>
int ldapssl_clientauth_init( char *certdbpath, void *certdbhandle,
                             int needkeydb, char *keydbpath, void *keydbhandle );
```

Parameters

This function has the parameters listed in Table 17-179.

Table 17-179 `ldapssl_clientauth_init()` Function Parameters

Parameter	Description
<code>certdbpath</code>	Specifies the path to the database containing certificates for your client. You can include the database filename in the path (for example, <code>/usr/mozilla/cert_file_name.db</code>).
<code>needkeydb</code>	Specifies whether or not the private key database needs to be opened for use. This parameter can have one of the following values: - If it is a non-zero value, the function opens the private key database, which is identified by the <code>keydbpath</code> argument. - If 0, the function does not open the private key database.
<code>keydbpath</code>	Specifies the path to the database containing the private key certified by your certificate. You can include the database filename in the path (for example, <code>/usr/mozilla/key_file_name.db</code>).
<code>certdbhandle</code>	Pass a NULL value for this. (This parameter is not currently used.)

Returns

One of the following values:

- 0 if successful.
- 1 if unsuccessful.

Description

You can call the `ldapssl_clientauth_init()` function to initialize your client application for SSL and certificate-based client authentication. However, if you need to specify the name and path of the security modules database or if you need to specify how the server's certificate will be verified, you should call the `ldapssl_advclientauth_init()` function instead.

NOTE

You must call `ldapssl_clientauth_init()` before calling `ldapssl_init()` to connect to the server. For details, see Chapter 11, "Connecting Over SSL."

Example

Code Example 17-56 initializes a client before connecting with a secure LDAP server.

Code Example 17-56 `ldapssl_clientauth_init()` Code Example

```
#include <ldap.h>
#include <ldap_ssl.h>
#include <stdio.h>
...
/* Initialize client, using mozilla's certificate database */
if ( ldapssl_clientauth_init( "/u/mozilla/.netscape/cert_file_name.db", NULL, 1,
    "/u/mozilla/.netscape/key_file_name.db", NULL ) < 0 ) {
    perror( "ldap_clientauth_init" );
    return( 1 );
}
...
}
```

See Also

`ldap_init()`, `ldapssl_init()`, `ldapssl_install_routines()`

ldapssl_enable_clientauth()

The `ldapssl_enable_clientauth()` function enables SSL client authentication on the given connection (passed using the `ld` parameter).

Syntax

```
#include <ldap_ssl.h>
int ldapssl_enable_clientauth( LDAP *ld, char *keynickname,
    char *keypasswd, char *certnickname );
```


Parameters

This function has the parameters listed in Table 17-180.

Table 17-180 ldapssl_enable_clientauth() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
keynickname	Pass an empty string ("") for this value. (This parameter is not currently used.)
keypasswd	Password to the encrypted private key database.
certnickname	Nickname of the certificate that you want to use for client authentication.

Returns

One of the following values:

- 0 if successful.
- -1 if unsuccessful.

See Also

ldapssl_clientauth_init()

ldapssl_err2string()

The ldapssl_err2string() function returns the corresponding error message for an SSL-specific error code.

Syntax

```
#include <ldap_ssl.h>
const char * LDAP_CALL ldapssl_err2string ( const int prerrno );
```

Parameters

This function has the parameters listed in Table 17-181.

Table 17-181 ldapssl_err2string() Function Parameters

Parameter	Description
prerrno	The SSL error code that you want interpreted into an error message.

Returns

One of the following values:

- If successful, returns the corresponding error message for the given error code.
- If unsuccessful (for example, if the error code is not a known SSL error code), returns a pointer to the string `Unknown error`.

Description

`ldapssl_err2string()` provides support for SSL-specific error messages that are not covered by the regular message routine `ldap_err2string()`. If any `ldapssl_*` function returns an error code that is unknown to `ldap_err2string()` (it returns `Unknown error`), this function should be called to determine the SSL-specific error message. To check for SSL errors, call `ldapssl_err2string()` after you call any of the following SSL initialization functions:

- `ldapssl_client_init()`
- `ldapssl_clientauth_init()`
- `ldapssl_advclientauth_init()`
- `ldapssl_pkcs_init()`

The errors returned by these functions are usually related to certificate database corruption, missing certs in a certificate database, client authentication failures, and other general SSL errors.

For more information, see “Getting the Error Message from an Error Code” on page 91 of Chapter 4, “Using the LDAP API.”

See Also

`ldapssl_client_init()`, `ldapssl_clientauth_init()`,
`ldapssl_advclientauth_init()`, `ldapssl_pkcs_init()`, `ldap_err2string()`

ldapssl_init()

The `ldapssl_init()` function initializes the LDAP library for SSL and installs the I/O routines for SSL.

Syntax

```
#include <ldap_ssl.h>
LDAP *ldapssl_init( const char *defhost, int defport, int defsecure );
```

Parameters

This function has the parameters listed in Table 17-182.

Table 17-182 ldapssl_init() Function Parameters

Parameter	Description
defhost	Connect to this LDAP server, if no other server is specified.
defport	Connect to this server port, if no other port is specified. To specify the default port 389, use LDAP_PORT as the value for this parameter.
defsecure	Determines whether or not to establish the default connection over SSL. Set this to a non-zero value to establish the default connection over SSL.

Returns

One of the following values:

- If successful, returns a pointer to an LDAP structure, which should be passed to subsequent calls to other LDAP API functions.
- If unsuccessful, returns a -1.

Description

ldapssl_init() allocates an LDAP structure but does not open an initial connection. Before calling this function, call ldapssl_client_init() to initialize your client for SSL. For details, see Chapter 11, “Connecting Over SSL.”

NOTE

Calling this function is equivalent to calling ldap_init() followed by ldapssl_install_routines() and ldap_set_option() to set the LDAP_OPT_SSL option to LDAP_OPT_ON.

Example

Code Example 17-57 connects your client to a secure LDAP server.

Code Example 17-57 ldapssl_init() Code Example

```
#include <ldap.h>
#include <ldap_ssl.h>
#include <stdio.h>
...
/* Initialize client, using mozilla's certificate database */
if ( ldapssl_client_init( "/u/mozilla/.netscape/cert_file_name.db", NULL ) < 0)
{
    printf( "Failed to initialize SSL client...\n" );
    return( 1 );
}
/* get a handle to an LDAP connection */
```

Code Example 17-57 ldapssl_init() Code Example (*Continued*)

```
if ( (ld = ldapssl_init( "cert.netscape.com", LDAPS_PORT, 1 )) == NULL {
    perror( "ldapssl_init" );
    return( 1 );
}
...
/* Client can now perform LDAP operations on the secure LDAP server */
...
```

See Also

ldap_init(), ldapssl_client_init(), ldapssl_install_routines()

ldapssl_install_routines()

The ldapssl_install_routines() function installs the I/O routines that enable SSL over LDAP.

Syntax

```
#include <ldap_ssl.h>
int ldapssl_install_routines( LDAP *ld );
```

Parameters

This function has the parameter listed in Table 17-183.

Table 17-183 ldapssl_install_routines() Function Parameter

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.

Returns

One of the following values:

- 0 if successful.
- -1 if unsuccessful.

Description

You need to call `ldapssl_install_routines()` in combination with `ldap_init()` and `ldap_set_option()`. As an alternative, you can call `ldapssl_init()` rather than these three functions.

NOTE

As is the case with the `ldapssl_init()` function, you need to call `ldapssl_client_init()` to initialize your client for SSL before calling `ldapssl_install_routines()`.

Example

Code Example 17-58 connects your client to a secure LDAP server.

Code Example 17-58 `ldapssl_install_routines()` Code Example

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

/*
 * Initialize client, using mozilla's certificate database */
if ( ldapssl_client_init( "/u/mozilla/.netscape/cert_file_name.db", NULL ) < 0 )
{
    printf( "Failed to initialize SSL client...\n" );
    return( 1 );
}

/* Get the handle to an LDAP connection */
if ( (ld = ldap_init( MY_HOST, 6360 )) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
}

/* Load SSL routines */
if ( ldapssl_install_routines( ld ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldapssl_install_routines" );
    return( 1 );
}

/* Set the option to use SSL with the default connection */
if (ldap_set_option( ld, LDAP_OPT_SSL, LDAP_OPT_ON ) != LDAP_SUCCESS) {
    ldap_perror( ld, "ldap_set_option" );
    return( 1 );
}
...
```

See Also

`ldap_init()`, `ldapssl_init()`, `ldapssl_client_init()`

ldapssl_pkcs_init()

CAUTION This function is deprecated and should not be used. It is included in `ldap-deprecated.h` for backward-compatibility.

The `ldap_pkcs_init()` function provides thread-safe SSL initialization.

Returns

One of the following values:

- 0 is successful.
- -1 if unsuccessful.
- *n* (a positive integer) denotes a Netscape Portable Runtime (NSPR) error as returned by the `PR_GetError()` NSPR function. Consult the NSPR documentation at <http://www.mozilla.org/projects/nspr/> for more information.

Description

The `ldap_pkcs_init()` structure sets up callbacks for the security library to obtain required runtime information. It should be used in place of `ldapssl_client_init()`, `ldapssl_clientauth_init()`, and `ldapssl_advclientauth_init()`.

NOTE The LDAP SDK for C uses the Public Key Cryptography Standard (PKCS) API implemented in Network Security Services (NSS) to provide SSL security support. Specifically, NSS implements the security API as defined in the PKCS#11 standard.

Because `ldap_pkcs_init()` is based on the `ldapssl_pkcs_fns` structure, you do not need to know all of the security parameters at initialization, unlike the other SSL initialization functions (`ldapssl*_init()`), which require all security parameters to be known at the time of initialization. Code Example 17-59 defines the `ldapssl_pkcs_fns` structure.

Code Example 17-59 ldapssl_pkcs_fns Structure Definition

```
typedef int (LDAP_PKCS_GET_TOKEN_CALLBACK)
(void *context, char **tokenname);
typedef int (LDAP_PKCS_GET_PIN_CALLBACK)
(void *context, const char *tokenname, char **tokenpin);
typedef int (LDAP_PKCS_GET_CERTPATH_CALLBACK)
```

Code Example 17-59 ldapssl_pkcs_fns Structure Definition (*Continued*)

```

    (void *context, char **certpath);
typedef int (LDAP_PKCS_GET_KEYPATH_CALLBACK)(void *context,
    char **keypath);
typedef int (LDAP_PKCS_GET_MODPATH_CALLBACK)
    (void *context, char **modulepath);
typedef int (LDAP_PKCS_GET_CERTNAME_CALLBACK)
    (void *context, char **certname);
typedef int (LDAP_PKCS_GET_DONGLEFILENAME_CALLBACK)
    (void *context, char **filename);

#define PKCS_STRUCTURE_ID 1
struct ldapssl_pkcs_fns {
    int local_structure_id;
    void *local_data;
    LDAP_PKCS_GET_CERTPATH_CALLBACK *pkcs_getcertpath;
    LDAP_PKCS_GET_CERTNAME_CALLBACK *pkcs_getcertname;
    LDAP_PKCS_GET_KEYPATH_CALLBACK *pkcs_getkeypath;
    LDAP_PKCS_GET_MODPATH_CALLBACK *pkcs_getmodpath;
    LDAP_PKCS_GET_PIN_CALLBACK *pkcs_getpin;
    LDAP_PKCS_GET_TOKEN_CALLBACK *pkcs_gettokenname;
    LDAP_PKCS_GET_DONGLEFILENAME_CALLBACK *pkcs_getdonglefilename;
};

```

See Also

ldapssl_client_init(), ldapssl_clientauth_init(),
 ldapssl_advclientauth_init()

ldapssl_serverauth_init()

The ldapssl_serverauth_init() function is a server-authentication only version of ldapssl_clientauth_init().

Syntax

```

#include <ldap_ssl.h>
int ldapssl_serverauth_init(
    const char *certdbpath, void *certdbhandle, const int sslstrength );

```

Parameters

This function has the parameters listed in Table 17-184.

NOTE

Please see the description of the sslstrength value and note the potential problems that can be caused by passing in wrong host and portname values.

Table 17-184 ldapssl_serverauth_init() Function Parameters

Parameter	Description
certdbpath	Specifies the path to the database containing certificates for your client. You can include the database filename in the path (for example, /usr/mozilla/cert_file_name.db).
certdbhandle	Pass a NULL value for this. (This parameter is not currently used.)
sslstrength	Specifies how the server certificate is evaluated. It takes one of the following: • LDAPSSL_AUTH_WEAK indicates that you accept the server's certificate without checking the for certificate authority (CA) that issued the certificate. • LDAPSSL_AUTH_CERT indicates that you accept the server's certificate only if you trust the CA that issued the certificate. • LDAPSSL_AUTH_CNCHECK indicates that you accept the server's certificate only if you trust the CA that issued the certificate and if the value of the cn attribute is the DNS hostname of the server. If this option is selected, please ensure that the defhost parameter passed to ldapssl_init() consists of only one hostname and not a list of hosts. Furthermore, the port number must be passed via the defport parameter, and cannot be passed via a host:port option.

Returns

One of the following values:

- 0 if successful.
- -1 if unsuccessful.

Description

ldapssl_serverauth_init() is a server-authentication only version of ldapssl_clientauth_init(). This function allows the sslstrength parameter to be passed in.

See Also

ldap_init(), ldapssl_init(), ldapssl_install_routines(),
ldapssl_clientauth_init()

ldapssl_set_strength()

The ldapssl_set_strength() sets the SSL strength for an existing SSL-enabled LDAP session handle.

Syntax

```
#include <ldap_ssl.h>
int LDAP_CALL ldapssl_set_strength( LDAP *ld, int sslstrength );
```

Parameters

This function has the parameters listed in Table 17-185.

NOTE

Please see the description of the `sslstrength` value and note the potential problems that can be caused by passing in wrong host and portname values.

Table 17-185 ldapssl_set_strength() Function Parameters

Parameter	Description
<code>ld</code>	Pointer to an LDAP structure containing information about the connection to the LDAP server. If value is NULL, the default for the new LDAP session handles is set.
<code>sslstrength</code>	Specifies how the server certificate is evaluated. It takes one of the following: - LDAPSSL_AUTH_WEAK indicates that you accept the server's certificate without checking the for certificate authority (CA) that issued the certificate. - LDAPSSL_AUTH_CERT indicates that you accept the server's certificate only if you trust the CA that issued the certificate. - LDAPSSL_AUTH_CNCHECK indicates that you accept the server's certificate only if you trust the CA that issued the certificate and if the value of the <code>cn</code> attribute is the DNS hostname of the server. If this option is selected, please ensure that the <code>defhost</code> parameter passed to <code>ldapssl_init()</code> consists of only one hostname and not a list of hosts. Furthermore, the port number must be passed via the <code>defport</code> parameter, and cannot be passed via a <i>host:port</i> option.

ldapssl_tls_start_s()

The `ldapssl_tls_start_s()` function starts an SSL handshake on a previously established non-secure connection.

Syntax

```
#include <ldap_ssl.h>
LDAP_API(int) LDAP_CALL ldapssl_tls_start_s( LDAP *ld, int defsecure,
char *certdbpath, char *keydbpath, char ***referrals );
```

Parameters

This function has the parameter listed in Table 17-186.

Table 17-186 ldapssl_tls_start_s() Function Parameter

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
defsecure	Determines whether or not to establish the default connection over SSL. Set this to a non-zero value to establish the default connection over SSL.
certdbpath	Specifies the path to the database containing certificates for your client. You can include the database filename in the path (for example, /usr/mozilla/cert_file_name.db).
keydbpath	Specifies the path to the database containing the private key certified by your certificate. You can include the database filename in the path (for example, /usr/mozilla/key_file_name.db).
referralsp	Pointer to an array of strings representing the referrals found by an LDAP search operation and returned by the server (applicable only if the LDAP operation was a search operation). When done, you can free this by calling the ldap_value_free() function.

Description

The ldapssl_tls_start_s() function starts an SSL handshake on a previously established non-secure connection.

ldap_str2charray()

This function is deprecated and should not be used. It is included in ldap-deprecated.h for backward-compatibility.

ldap_tmplattrs()

The ldap_tmplattrs() function obtains a pointer to the correct ldap_disptmpl structure.

Syntax

```
#include <disptmpl.h>
char ** ldap_tmplattrs( struct ldap_disptmpl *tmpl, char **includeattrs,
    int exclude, unsigned long syntaxmask );
```

Parameters

This function has the parameters listed in Table 17-187.

Table 17-187 ldap_tmplattr() Function Parameters

Parameter	Description
tmpl	The name of the template to be retrieved.
includeattrs	A null-terminated array of attributes that should always be included (it may be NULL if no extra attributes are required).
exclude	If 0, only attributes where the logical AND of the template item syntax id and the syntaxmask is non-zero are included. If non-zero, attributes where the logical AND of the template item syntax id and the syntaxmask is non-zero are excluded.
syntaxmask	When non-zero, it is used to restrict the attribute set returned.

Returns

- A NULL-terminated array that contains the names of attributes that need to be retrieved if the template, defined by `tmpl`, is to be used to display an entry.
- A NULL pointer on error.

Description

The attribute list should be freed using `ldap_value_free()`.

See Also

`ldap_value_free()`

ldap_tmplerr2string()

The `ldap_tmplerr2string()` function returns a string representation of the error passed in the parameter.

Syntax

```
#include <disptmpl.h>
char * ldap_tmplerr2string( int err );
```

Parameters

This function has the parameter listed in Table 17-188.

Table 17-188 ldap_tmplerr2string() Function Parameter

Parameter	Description
err	Error returned.

ldap_ufn_search_c()

CAUTION This function will be deprecated and should not be used. It is included in ldap-to-be-deprecated.h for backward-compatibility.

ldap_ufn_search_ct()

CAUTION This function will be deprecated and should not be used. It is included in ldap-to-be-deprecated.h for backward-compatibility.

ldap_ufn_search_s()

CAUTION This function will be deprecated and should not be used. It is included in ldap-to-be-deprecated.h for backward-compatibility.

ldap_ufn_setfilter()

CAUTION This function will be deprecated and should not be used. It is included in ldap-to-be-deprecated.h for backward-compatibility.

ldap_ufn_setprefix()

CAUTION This function will be deprecated and should not be used. It is included in ldap-to-be-deprecated.h for backward-compatibility.

ldap_ufn_timeout()

CAUTION This function will be deprecated and should not be used. It is included in `ldap-to-be-deprecated.h` for backward-compatibility.

ldap_unbind()

The `ldap_unbind()` function unbinds from the directory, terminates the current association, and frees the resources contained in the LDAP structure.

Syntax

```
#include <ldap.h>
int ldap_unbind( LDAP *ld );
```

Parameters

This function has the parameter listed in Table 17-189.

Table 17-189 ldap_unbind() Function Parameter

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.

Returns

For a list of possible results for an LDAP unbind operation, see `ldap_unbind_s()`.

Description

The `ldap_unbind()` function unbinds from the directory, terminates the current association, and frees the resources contained in the LDAP structure. The three unbind functions (`ldap_unbind()`, `ldap_unbind_s()`, and `ldap_unbind_ext()`) all work synchronously 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

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 calls using it.

For additional information, see “Closing the Connection to the Server” on page 73 of Chapter 3, “Writing an LDAP Client.”

Example

Code Example 17-60 closes the current connection with the LDAP server.

Code Example 17-60 ldap_unbind() Code Example

```
#include <ldap.h>
...
LDAP *ld;
...
/* After completing your LDAP operations with the server, close the
connection. */
if ( ldap_unbind( ld ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "Error while unbinding from the directory" );
    return( 1 );
}
...
```

See Also

ldap_unbind_s(), ldap_unbind_ext()

ldap_unbind_s()

The ldap_unbind_s() function unbinds from the directory, terminates the current association, and frees the resources contained in the LDAP structure.

Syntax

```
#include <ldap.h>
int ldap_unbind_s( LDAP *ld );
```

Parameters

This function has the parameter listed in Table 17-190.

Table 17-190 ldap_unbind_s() Function Parameter

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the request.
- LDAP_SERVER_DOWN if the LDAP server did not receive the request or if the connection to the server was lost.
- LDAP_NO_MEMORY if memory cannot be allocated.

Description

The `ldap_unbind_s()` function unbinds from the directory, terminates the current association, and frees the resources contained in the LDAP structure. The three unbind functions (`ldap_unbind()`, `ldap_unbind_s()`, and `ldap_unbind_ext()`) all work synchronously 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

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 calls using it.

For additional information, see “Closing the Connection to the Server” on page 73 of Chapter 3, “Writing an LDAP Client.”

Example

Code Example 17-61 closes the current connection with the LDAP server.

Code Example 17-61 ldap_unbind_s() Code Example

```
#include <ldap.h>
...
LDAP *ld;
...
/* After completing your LDAP operations with the server, close the
connection. */
if ( ldap_unbind_s( ld ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "Error while unbinding from the directory" );
    return( 1 );
}
...
```

See Also

ldap_unbind(), ldap_unbind_ext()

ldap_unbind_ext()

The ldap_unbind_ext() function unbinds from the directory, terminates the current association, and frees the resources contained in the LDAP structure.

Syntax

```
#include <ldap.h>
int ldap_unbind_ext( LDAP *ld, LDAPControl **serverctrls,
    LDAPControl **clientctrls );
```

Parameters

This function has the parameters listed in Table 17-191.

Table 17-191 ldap_unbind_ext() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
serverctrls	List of LDAP server controls.
clientctrls	List of client controls.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the request.
- LDAP_SERVER_DOWN if the LDAP server did not receive the request or if the connection to the server was lost.
- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_NOT_SUPPORTED if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see “Specifying the LDAP Version of Your Client” on page 65 of Chapter 3, “Writing an LDAP Client.”)

Description

The `ldap_unbind_ext()` function unbinds from the directory, terminates the current association, and frees the resources contained in the LDAP structure. The three unbind functions (`ldap_unbind()`, `ldap_unbind_s()`, and `ldap_unbind_ext()`) all work synchronously 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

Unlike the other two unbind functions, `ldap_unbind_ext()` allows you to explicitly include both server and client controls in your unbind request. However, since there is no server response to an unbind request, there is no way to receive a response from a server control that is included.

For additional information, see “Closing the Connection to the Server” on page 73 of Chapter 3, “Writing an LDAP Client.”

See Also

`ldap_unbind()`, `ldap_unbind_s()`

ldap_url_parse()

The `ldap_url_parse()` function parses an LDAP URL into its components.

Syntax

```
#include <ldap.h>
int ldap_url_parse( const char *url, LDAPURLDesc **ludpp );
```

Parameters

This function has the parameters listed in Table 17-192.

Table 17-192 `ldap_url_parse()` Function Parameters

Parameter	Description
<code>url</code>	The URL that you want to check.
<code>ludpp</code>	Pointer to a structure containing the components of the URL.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_URL_ERR_NOTLDAP if the URL does not begin with the ldap:// or ldaps:// prefix.
- LDAP_URL_ERR_NODN if the URL missing trailing slash after host or port.
- LDAP_URL_ERR_BADSCOPE if the scope within the URL is invalid.
- LDAP_URL_ERR_MEM if not enough free memory is available for this operation.
- LDAP_URL_ERR_PARAM if an invalid argument was passed to the function.

Description

For more information, see “Getting the Components of an LDAP URL” on page 203 in Chapter 9, “Working with LDAP URLs.”

Example

Code Example 17-62 parses an LDAP URL and prints out each component of it.

Code Example 17-62 ldap_url_parse() Code Example

```
#include <stdio.h>
#include <ldap.h>
...
char *my_url =
"ldap://ldap.sun.com:5000/dc=example,dc=com?cn,mail,telephoneNumber?sub?(sn
=Jensen)";
LDAPURLDesc *ludpp;
int res, i;
...
if ( ( res = ldap_url_parse( my_url, &ludpp ) ) != 0 ) {
    switch( res ){
        case LDAP_URL_ERR_NOTLDAP:
            printf( "URL does not begin with \"ldap://\"\\n" );
            break;
        case LDAP_URL_ERR_NODN:
            printf( "URL missing trailing slash after host or port\\n" );
            break;
        case LDAP_URL_ERR_BADSCOPE:
            printf( "URL contains an invalid scope\\n" );
            break;
        case LDAP_URL_ERR_MEM:
            printf( "Not enough memory\\n" );
            break;
        default:
            printf( "Unknown error\\n" );
    }
    return( 1 );
}
printf( "Components of the URL:\\n" );
printf( "Host name: %s\\n", ludpp->lud_host );
```

Code Example 17-62 ldap_url_parse() Code Example (*Continued*)

```

printf( "Port number: %d\n", ludpp->lud_port );
if ( ludpp->lud_dn != NULL ) {
    printf( "Base entry: %s\n", ludpp->lud_dn );
} else {
    printf( "Base entry: Root DN\n" );
}
if ( ludpp->lud_attrs != NULL ) {
    printf( "Attributes returned: \n" );
    for ( i=0; ludpp->lud_attrs[i] != NULL; i++ ) {
        printf( "\t%s\n", ludpp->lud_attrs[i] );
    }
} else {
    printf( "No attributes returned.\n" );
}
printf( "Scope of the search: " );
switch( ludpp->lud_scope ) {
    case LDAP_SCOPE_BASE:
        printf( "base\n" );
        break;
    case LDAP_SCOPE_ONELEVEL:
        printf( "one\n" );
        break;
    case LDAP_SCOPE_SUBTREE:
        printf( "sub\n" );
        break;
    default:
        printf( "Unknown scope\n" );
}
printf( "Filter: %s\n", ludpp->lud_filter );
...

```

See Also

ldap_free_urldesc()

ldap_url_parse_no_defaults()

The `ldap_url_parse_no_defaults()` function parses an LDAP URL into its components.

Syntax

```

#include <ldap.h>
int ldap_url_parse_no_defaults( const char *url, LDAPURLDesc **ludpp,
    int dn_required );

```

Parameters

This function has the parameters listed in Table 17-193.

Table 17-193 ldap_url_parse_no_defaults() Function Parameters

Parameter	Description
url	The URL that you want to check.
ludpp	Pointer to a structure containing the components of the URL.
dn_required	Specifies if a DN must be present in the URL. Set to 0, the DN is not required. Set to 1, the function will return an error LDAP_URL_ERR_NODN if no DN is present.

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_URL_ERR_NODN if a required DN is not present.
- LDAP_URL_ERR_NOTLDAP if the URL does not begin with the ldap:// or ldaps:// prefix.
- LDAP_URL_ERR_NODN if the URL missing trailing slash after host or port.
- LDAP_URL_ERR_BADSCOPE if the scope within the URL is invalid.
- LDAP_URL_ERR_MEM if not enough free memory is available for this operation.
- LDAP_URL_ERR_PARAM if an invalid argument was passed to the function.

Description

The ldap_url_parse_no_defaults() function does not set any default value in the fields that would be absent from the URL (for example, the port number).

ldap_url_search()

The ldap_url_search() function searches the directory asynchronously for matching entries, based on the contents of the URL.

Syntax

```
#include <ldap.h>
int ldap_url_search( LDAP *ld, const char *url, int attrsonly );
```

Parameters

This function has the parameters listed in Table 17-194.

Table 17-194 ldap_url_search() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
url	LDAP URL specifying a search of the directory.
attrsonly	Specifies whether or not attribute values are returned along with the attribute types. This parameter can have the following values: 0 specifies that both attribute types and attribute values are returned. 1 specifies that only attribute types are returned.

Returns

Returns the message ID of the ldap_url_search() operation. To check the result of this operation, call ldap_result() and ldap_result2error().

Description

ldap_url_search() searches the directory for matching entries, based on the contents of the URL. ldap_url_search() is an asynchronous function; it does not directly return results. If you want the results to be returned directly by the function, call the synchronous function ldap_url_search_s().

For more information on processing LDAP searches specified as URLs, see “Processing an LDAP URL” on page 207 of Chapter 9, “Working with LDAP URLs.”

Example

Code Example 17-63 returns the message ID.

Code Example 17-63 ldap_url_search() Code Example

```
#include "examples.h"

static void do_other_work();
unsigned long global_counter = 0;

int
main( int argc, char **argv )
{
    char *my_url =
"ldap://ldap.example.com/dc=example,dc=com?cn,mail,telephoneNumber?sub?(sn=
Jensen)";
    LDAP *ld;
    LDAPMessage *result, *e;
```

Code Example 17-63 ldap_url_search() Code Example (*Continued*)

```

BerElement *ber;
char *a, *dn;
char **vals;
int i, rc, finished, msgid;
int num_entries = 0;
struct timeval zerotime;

zerotime.tv_sec = zerotime.tv_usec = 0L;

/* get a handle to an LDAP connection */
if ( ( ld = ldap_init( MY_HOST, MY_PORT ) ) == NULL ) {
    perror( "ldap_init" );
    return( 1 );
}
/* authenticate to the directory as nobody */
if ( ldap_simple_bind_s( ld, NULL, NULL ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_simple_bind_s" );
    return( 1 );
}
/* search for all entries with surname of Jensen */
if ( ( msgid = ldap_url_search( ld, my_url, 0 ) ) == -1 ) {
    ldap_perror( ld, "ldap_url_search" );
    return( 1 );
}

/* Loop, polling for results until finished */
finished = 0;
while ( !finished ) {
    /*
     * Poll for results. We call ldap_result with the "all" parameter
     * set to zero. This causes ldap_result() to return exactly one
     * entry if at least one entry is available. This allows us to
     * display the entries as they are received.
     */
    result = NULL;
    rc = ldap_result( ld, msgid, 0, &zerotime, &result );
    switch ( rc ) {
        case -1:
            /* some error occurred */
            ldap_perror( ld, "ldap_result" );
            return( 1 );
        case 0:
            /* Timeout was exceeded. No entries are ready for retrieval. */
            if ( result != NULL ) {
                ldap_msgfree( result );
            }
            break;
        default:
            /*
             * Either an entry is ready for retrieval, or all entries have
             * been retrieved.
             */
            if ( ( e = ldap_first_entry( ld, result ) ) == NULL ) {
                /* All done */
                finished = 1;
            }
    }
}

```

Code Example 17-63 ldap_url_search() Code Example (*Continued*)

```

        if ( result != NULL ) {
            ldap_msgfree( result );
        }
        continue;
    }
    /* for each entry print out name + all attrs and values */
    num_entries++;
    if (( dn = ldap_get_dn( ld, e )) != NULL ) {
        printf( "dn: %s\n", dn );
        ldap_memfree( dn );
    }
    for ( a = ldap_first_attribute( ld, e, &ber );
          a != NULL; a = ldap_next_attribute( ld, e, ber ) ) {
        if (( vals = ldap_get_values( ld, e, a )) != NULL ) {
            for ( i = 0; vals[ i ] != NULL; i++ ) {
                printf( "%s: %s\n", a, vals[ i ] );
            }
            ldap_value_free( vals );
        }
        ldap_memfree( a );
    }
    if ( ber != NULL ) {
        ldap_ber_free( ber, 0 );
    }
    printf( "\n" );
    ldap_msgfree( result );
}
/* Do other work here while you are waiting... */
do_other_work();
}

/* All done. Print a summary. */
printf( "%d entries retrieved. I counted to %ld "
        "while I was waiting.\n", num_entries,
        global_counter );
ldap_unbind( ld );
return( 0 );
}

/*
 * Perform other work while polling for results. */
static void
do_other_work()
{
    global_counter++;
}

```

See Also

ldap_url_search_s(), ldap_result(), ldap_result2error()

ldap_url_search_s()

The `ldap_url_search_s()` function searches the directory synchronously for matching entries, based on the contents of the URL.

Syntax

```
#include <ldap.h>
int ldap_url_search_s( LDAP *ld, const char *url,
    int attrsonly, LDAPMessage **res );
```

Parameters

This function has the parameters listed in Table 17-195.

Table 17-195 ldap_url_search_s() Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
url	LDAP URL specifying a search of the directory.
attrsonly	Specifies whether or not attribute values are returned along with the attribute types. This parameter can have the following values: • 0 specifies that both attribute types and attribute values are returned. • 1 specifies that only attribute types are returned.
res	Results of the search (when the call is completed).

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- If unsuccessful, returns the LDAP error code for the operation.

See Chapter 18, “Result Codes” for a complete listing.

Description

The `ldap_url_search_s()` function searches the directory for matching entries, based on the contents of the URL. `ldap_url_search_s()` is a synchronous function, which directly returns the results of the operation. If you want to perform other operations while waiting for the results of this operation, call the asynchronous function `ldap_url_search()`.

For more information on processing LDAP searches specified as URLs, see “Processing an LDAP URL” on page 207 of Chapter 9, “Working with LDAP URLs.”

Example

Code Example 17-64 processes a search request from an LDAP URL.

Code Example 17-64 ldap_url_search_s() Code Example

```
#include <stdio.h>
#include <ldap.h>
...
LDAP *ld;
LDAPMessage *result;
char *my_url =
"ldap://ldap.sun.com/dc=example,dc=com?cn,mail,telephoneNumber?sub?(sn=Jens
en)";
...
/* Process the search request in the URL */
if ( ldap_url_search_s( ld, my_url, 0, &result ) != LDAP_SUCCESS ) {
    ldap_perror( ld, "ldap_url_search_s" );
    return( 1 );
}
...
```

See Also

ldap_search(), ldap_search_st()

ldap_url_search_st()

The `ldap_url_search_st()` function searches the directory, synchronously within a specified time limit, for matching entries, based on the contents of the URL.

Syntax

```
#include <ldap.h>
int ldap_url_search_st( LDAP *ld, const char *url, int attrsonly,
    struct timeval *timeout, LDAPMessage **res );
```

Parameters

This function has the parameters listed in Table 17-196.

Table 17-196 ldap_url_search_st()Function Parameters

Parameter	Description
ld	Pointer to an LDAP structure containing information about the connection to the LDAP server.
url	LDAP URL specifying a search of the directory.
attrsonly	Specifies whether or not attribute values are returned along with the attribute types. This parameter can have the following values: • 0 specifies that both attribute types and attribute values are returned. • 1 specifies that only attribute types are returned.
timeout	Maximum time to wait for the results of the search.
res	Results of the search (when the call is completed).

Returns

One of the following values:

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR if any of the arguments are invalid.
- LDAP_ENCODING_ERROR if an error occurred when BER-encoding the request.
- LDAP_SERVER_DOWN if the LDAP server did not receive the request or if the connection to the server was lost.
- LDAP_NO_MEMORY if memory cannot be allocated.
- LDAP_LOCAL_ERROR if an error occurred when receiving the results from the server.
- LDAP_DECODING_ERROR if an error occurred when decoding the BER-encoded results from the server.
- LDAP_FILTER_ERROR if an error occurred when parsing and BER-encoding the search filter specified by the filter argument.
- LDAP_TIMEOUT if the search exceeded the time specified by the timeoutp argument.

- LDAP_NOT_SUPPORTED if controls are included in your request (for example, as a session preference) and your LDAP client does not specify that it is using the LDAPv3. Make sure that you set the version of your LDAP client to version 3 before calling this function. (For details, see “Specifying the LDAP Version of Your Client” on page 65 of Chapter 3, “Writing an LDAP Client.”)

NOTE Sun Java System Directory Server and other LDAP server products may send result codes in addition to those described. For example, the server may have loaded a plug-in that returns custom result codes. Check your LDAP server documentation for other result codes.

Description
 The ldap_url_search_st() function searches the directory for matching entries, based on the contents of the URL. This function works like ldap_url_search_s() and lets you specify a timeout period for the search.

For more information, see “Processing an LDAP URL” on page 207 of Chapter 9, “Working with LDAP URLs.”

See Also
 ldap_search(), ldap_search_s()

ldap_utf8characters()

The ldap_utf8characters() function accepts a Unicode string and returns the number of characters.

Syntax

```
#include <ldap.h>
size_t ldap_utf8characters( const char* );
```

Parameters
 This function has the parameter listed in Table 17-197.

Table 17-197 ldap_utf8characters()Function Parameter

Parameter	Description
char	Pointer to a UTF-8 string.

Returns

Returns the number of UTF-8 characters in the 0-terminated array.

ldap_utf8copy()

The ldap_utf8copy() function accepts a Unicode string and copies the characters to a destination.

Syntax

```
#include <ldap.h>
int ldap_utf8copy( char* dst, const char* src );
```

Parameters

This function has the parameter listed in Table 17-198.

Table 17-198 ldap_utf8copy()Function Parameter

Parameter	Description
dst	Pointer to a destination for the copied characters.
src	Pointer to a UTF-8 string to copy.

Description

ldap_utf8copy() copies a character from src to dst. This function handles any valid UTF-8 character (including \0 and ASCII).

Returns

The number of characters copied.

ldap_utf8getcc()

The ldap_utf8getcc() function gets one UCS-4 character and moves the pointer to the next character.

Syntax

```
#include <ldap.h>
unsigned long ldap_utf8getcc( const char** src );
```

Parameters

This function has the parameter listed in Table 17-199.

Table 17-199 ldap_utf8getcc()Function Parameter

Parameter	Description
src	Pointer to a UCS-4 character.

ldap_utf8isalnum()

The `ldap_utf8isalnum()` function determines whether the character is an alphanumeric one.

Syntax

```
#include <ldap.h>
int ldap_utf8isalnum( char* s );
```

Parameters

This function has the parameter listed in Table 17-200.

Table 17-200 ldap_utf8isalnum()Function Parameter

Parameter	Description
s	Pointer to a UTF-8 character.

ldap_utf8isalpha()

The `ldap_utf8isalpha()` function determines whether the character is a letter.

Syntax

```
#include <ldap.h>
int ldap_utf8isalpha( char* s );
```

Parameters

This function has the parameter listed in Table 17-201.

Table 17-201 ldap_utf8isalpha()Function Parameter

Parameter	Description
s	Pointer to a UTF-8 character.

ldap_utf8isdigit()

The ldap_utf8isdigit() function determines whether the character is a number.

Syntax

```
#include <ldap.h>
int ldap_utf8isdigit( char* s );
```

Parameters

This function has the parameter listed in Table 17-202.

Table 17-202 ldap_utf8isdigit()Function Parameter

Parameter	Description
s	Pointer to a UTF-8 character.

ldap_utf8isspace()

The ldap_utf8isspace() function determines whether the character is a space, tab, newline, return or formfeed.

Syntax

```
#include <ldap.h>
int ldap_utf8isspace( char* s );
```

Parameters

This function has the parameter listed in Table 17-203.

Table 17-203 ldap_utf8isspace()Function Parameter

Parameter	Description
s	Pointer to a UTF-8 character.

ldap_utf8len()

The ldap_utf8len() function accepts a Unicode string and returns the number of bytes it contains.

Syntax

```
#include <ldap.h>
int ldap_utf8len( const char* );
```

Parameters

This function has the parameter listed in Table 17-204.

Table 17-204 ldap_utf8len()Function Parameter

Parameter	Description
char	Pointer to a UTF-8 string.

ldap_utf8next()

The ldap_utf8next() function accepts a character and returns a pointer to the character immediately following it.

Syntax

```
#include <ldap.h>
char* ldap_utf8next( char* );
```

Parameters

This function has the parameter listed in Table 17-205.

Table 17-205 ldap_utf8next()Function Parameter

Parameter	Description
char	Pointer to a UTF-8 character.

ldap_utf8prev()

The ldap_utf8prev() function accepts a character and returns a pointer to the previous character.

Syntax

```
#include <ldap.h>
char* ldap_utf8prev( char* );
```

Parameters

This function has the parameter listed in Table 17-206.

Table 17-206 ldap_utf8prev()Function Parameter

Parameter	Description
char	Pointer to a UTF-8 character.

ldap_utf8strtok_r()

The `ldap_utf8strtok_r()` function gets the next token from a string.

Syntax

```
#include <ldap.h>
char* ldap_utf8strtok_r( char* src, const char* brk, char** next);
```

Parameters

This function has the parameter listed in Table 17-207.

Table 17-207 ldap_utf8strtok_r() Function Parameter

Parameter	Description
src	Pointer to a UTF-8 string from which to extract token.
brk	Points to a NULL-terminated set of delimiter characters.
next	Pointer to the next token.

Returns

Returns a pointer to the next token. If there are no remaining tokens, returns a NULL pointer.

ldap_vals2html()

The `ldap_vals2html()` function writes the HTML representation of a set of values.

Syntax

```
#include <disptmpl.h>
int ldap_vals2html( LDAP *ld, char *buf, char **vals, char *label,
    int labelwidth, unsigned long syntaxid, writeptype writeproc,
    void *writeparm, char *eol, int rdncount, char *urlprefix );
```


Parameters

This function has the parameters listed in Table 17-208.

Table 17-208 ldap_vals2html() Function Parameters

Parameter	Description
ld	The LDAP pointer obtained by a previous call to <code>ldap_open()</code> .
buf	
vals	A NULL-terminated list of values, usually obtained by a call to <code>ldap_get_values()</code> .
label	A string (usually a friendly form of an LDAP attribute name) shown next to the values.
labelwidth	Specifies the label margin (the number of blank spaces displayed to the left of the values). If zero is passed, a default label width is used.
syntaxid	Display template attribute syntax identifier for a list of pre-defined LDAP_SYNTAX_... values.
writeproc	writeproc function should be declared as: <pre>int writeproc(writeparm, p, len) void *writeparm; char *p; int len;</pre> where p is a pointer to text to be written and len is the length of the text. p is guaranteed to be zero-terminated.
writeparm	A pointer to a structure that will be passed as the first parameter of the writeproc procedure. Typically, this is used to pass the file descriptor of the file to write to.
eol	Lines of text are terminated with this string.
rdncount	Limits the number of components that are displayed for DN attributes.
urlprefix	Starting text to use when constructing an LDAP URL. The default is the string <code>ldap://</code>

Returns

- LDAP_SUCCESS if successful.
- An LDAP error code on error.

Description

`ldap_vals2html()` produces HTML output of a single set of LDAP attribute values.

See Also
ldap_vals2text()

ldap_vals2text()

The ldap_vals2text() function writes the text representation of an LDAP entry.

Syntax

```
#include <disptmpl.h>
int ldap_vals2text( LDAP *ld, char *buf, char **vals, char *label,
    int labelwidth, unsigned long syntaxid, writeptype writeproc,
    void *writeparm, char *eol, int rdncount );
```

Parameters
This function has the parameters listed in Table 17-209.

Table 17-209 ldap_vals2text() Function Parameters

Parameter	Description
ld	The LDAP pointer obtained by a previous call to ldap_open().
buf	
vals	A NULL-terminated list of values, usually obtained by a call to ldap_get_values().
label	A string (usually a friendly form of an LDAP attribute name) shown next to the values.
labelwidth	Specifies the label margin (the number of blank spaces displayed to the left of the values). If zero is passed, a default label width is used.
syntaxid	Display template attribute syntax identifier for a list of pre-defined LDAP_SYN_... values.
writeproc	writeproc function should be declared as: int writeproc(writeparm, p, len) void *writeparm; char *p; int len; where p is a pointer to text to be written and len is the length of the text. p is guaranteed to be zero-terminated.
writeparm	A pointer to a structure that will be passed as the first parameter of the writeproc procedure. Typically, this is used to pass the file descriptor of the file to write to.
eol	Lines of text are terminated with this string.

Table 17-209 ldap_vals2text() Function Parameters (*Continued*)

Parameter	Description
rdncount	Limits the number of components that are displayed for DN attributes.

Returns

- LDAP_SUCCESS if successful.
- An LDAP error code on error.

Description

ldap_vals2text() produces a text representation of a single set of LDAP attribute values.

See Also

ldap_vals2html()

ldap_value_free()

The ldap_value_free() function frees an array of values from memory.

Syntax

```
#include <ldap.h>
void ldap_value_free( char **values );
```

Parameters

This function has the parameter listed in Table 17-210.

Table 17-210 ldap_value_free() Function Parameter

Parameter	Description
values	Array of values.

Description

Use the ldap_value_free_len() function if the values are berval structures.

For additional information, see “Getting the Values of an Attribute” on page 128 of Chapter 5, “Searching a LDAP Directory.”

Example

See the example under `ldap_get_values()`.

See Also

`ldap_get_values()`, `ldap_value_free_len()`

ldap_value_free_len()

The `ldap_value_free_len()` function frees an array of `berval` structures from memory.

Syntax

```
#include <ldap.h>
void ldap_value_free_len( struct berval **values );
```

Parameters

This function has the parameters listed in Table 17-211.

Table 17-211 `ldap_value_free_len()` Function Parameter

Parameter	Description
values	Array of <code>berval</code> structures.

Description

Use the `ldap_value_free()` function instead of this function if the values are string values.

For additional information, see “Getting the Values of an Attribute” on page 128 of Chapter 5, “Searching a LDAP Directory.”

Example

See the example under `ldap_get_values_len()`.

See Also

`ldap_get_values()`, `ldap_get_values_len()`

ldap_version()

CAUTION This function is deprecated and should not be used. It is included in ldap-deprecated.h for backward-compatibility. Please use ldap_get_option() with LDAP_OPT_API_INFO and an LDAPAPIInfo structure.

The ldap_version() function gets version information about the LDAP SDK for C libraries.

Syntax

```
#include <ldap.h>
int ldap_version( LDAPVersion *ver );
```

Parameters

This function has the parameter listed in Table 17-212.

Table 17-212 ldap_version() Function Parameter

Parameter	Description
ver	LDAPVersion structure returning version information. If you only want the SDK version, you can pass NULL for this parameter.

Returns

The version number of the LDAP SDK for C, multiplied by 100. For example, for version 1.0, the function returns 100.

See Also

ldap_get_option()

ldap_x_calloc()

The ldap_x_calloc() function allocates space for an array of elements.

Syntax

```
#include <ldap-extension.h>
void *ldap_x_calloc( size_t nelem, size_t elsize );
```

Parameters

This function has the parameters listed in Table 17-214.

Table 17-213ldap_x_malloc() Function Parameters

Parameter	Description
nelem	Number of elements for which memory is to be allocated.
elsize	Size of each element.

Returns

- If successful, returns a pointer to the space.
- If there is no available memory, returns a NULL pointer.

See Also

ldap_x_free(), ldap_x_malloc(), ldap_x_realloc()

ldap_x_free()

The ldap_x_free() function frees allocated memory.

Syntax

```
#include <ldap-extension.h>
void ldap_x_free( void *ptr );
```

Parameters

This function has the parameter listed in Table 17-214.

Table 17-214ldap_x_free() Function Parameter

Parameter	Description
ptr	Pointer to the block of memory to be freed.

See Also

ldap_x_malloc(), ldap_x_malloc(), ldap_x_realloc()

ldap_x_hostlist_first()

The ldap_x_hostlist_first() function returns the first host and port defined in a host list file.

Syntax

```
#include <ldap-extension.h>
int ldap_x_hostlist_first( char *hostlist, int defport, char **hostp,
    int *portp, struct ldap_x_hostlist_status **statusp );
```

Description

ldap_x_hostlist_first() parses a space-separated host list (useful for implementing an extended I/O CONNECT callback function) and returns the first host and port defined.

Returns

- LDAP_SUCCESS if successful.
- If unsuccessful returns a valid LDAP API error code.

NOTE

A NULL or zero-length hostlist causes the host "127.0.0.1" to be returned.

See Also

ldap_x_hostlist_status, LDAP_X_EXTIOF_CONNECT_CALLBACK,

ldap_x_hostlist_next()

The ldap_x_hostlist_next() function returns the host and port defined in a host list file following the previous host and port returned.

Syntax

```
#include <ldap-extension.h>
int ldap_x_hostlist_next( char **hostp, int *portp,
    struct ldap_x_hostlist_status **statusp );
```

Description

This utility parses a space-separated host list (useful for implementing an extended I/O CONNECT callback function) and returns the host and port defined following the previous host and port returned.

Returns

- LDAP_SUCCESS if successful. (If no more hosts are available, LDAP_SUCCESS is returned but hostp is set to NULL.)
- If unsuccessful returns a valid LDAP API error code.

NOTE A NULL or zero-length hostlist causes the host "127.0.0.1" to be returned.

See Also
ldap_x_hostlist_status, LDAP_X_EXTIOF_CONNECT_CALLBACK,

ldap_x_hostlist_status()

See Also
ldap_x_hostlist_first(), ldap_x_hostlist_next(),
LDAP_X_EXTIOF_CONNECT_CALLBACK

ldap_x_hostlist_statusfree()

See Also
ldap_x_hostlist_first(), ldap_x_hostlist_next(),
LDAP_X_EXTIOF_CONNECT_CALLBACK

ldap_x_malloc()

The ldap_x_malloc() function allocates space for an object.

Syntax
#include <ldap-extension.h>
void *ldap_x_malloc(size_t size);

Parameters
This function has the parameter listed in Table 17-215.

Table 17-215ldap_x_malloc() Function Parameter

Parameter	Description
size	Specifies the size for the allocated block of memory.

See Also

ldap_x_free(), ldap_x_calloc(), ldap_x_realloc()

ldap_x_realloc()

The ldap_x_realloc() function changes the size of a block of memory.

Syntax

```
#include <ldap-extension.h>
void *ldap_x_realloc( void *ptr, size_t size );
```

Parameters

This function has the parameters listed in Table 17-216.

Table 17-216 ldap_x_realloc() Function Parameter

Parameter	Description
ptr	Pointer to a block of memory.
size	Specifies the new size of the block.

See Also

ldap_x_free(), ldap_x_calloc(), ldap_x_malloc()

prldap_get_default_socket_info()

The prldap_get_default_socket_info() function retrieves default socket information.

Syntax

```
#include <ldappr.h>
int prldap_get_default_socket_info( LDAP *ld, PRLDAPSocketInfo *soip );
```

Parameters

This function has the parameters listed in Table 17-217.

Table 17-217 prldap_get_default_socket_info() Function Parameters

Parameter	Description
fd	The session handle returned from prldap_init().

Table 17-217 prldap_get_default_socket_info() Function Parameters

Parameter	Description
soip	Pointer to a structure containing socket specific information.

Returns

- LDAP_SUCCESS if successful.
- An LDAP API error code if an error occurs.
- LDAP_PARAM_ERROR is returned if ld is NULL.

See Also

prldap_set_default_socket_info()

prldap_get_session_info()

The prldap_get_session_info() function retrieves application-specific data.

Syntax

```
#include <ldappr.h>
int prldap_get_session_info( LDAP *ld, void *sessionarg,
    PRLDAPSessionInfo *seip );
```

Parameters

This function has the parameters listed in Table 17-218.

Table 17-218 prldap_get_session_info() Function Parameters

Parameter	Description
ld	The session handle returned from prldap_init(). If NULL, the functions are installed as the default functions for all new LDAP handles.
seip	Pointer to a structure with session information data.

Returns

- LDAP_SUCCESS if successful.
- An LDAP API error code if an error occurs.

Description

The `prldap_get_session_option()` function retrieves an option for an LDAP session handle or a session argument that is passed to the `CONNECT`, `POLL`, `NEWHANDLE`, or `DISPOSEHANDLE` extended I/O callbacks.

See Also

`prldap_set_session_info()`

prldap_get_session_option()

The `prldap_get_session_option()` function retrieves an option specific to the `prldap` layer.

Syntax

```
#include <ldappr.h>
int prldap_get_session_option( LDAP *ld, void *sessionarg,
    int option, ... );
```

Returns

- `LDAP_SUCCESS` if successful.
- An LDAP API error code if an error occurs.

Description

The `prldap_get_session_option()` function retrieves an option for an LDAP session handle or a session argument that is passed to the `CONNECT`, `POLL`, `NEWHANDLE`, or `DISPOSEHANDLE` extended I/O callbacks.

See Also

`prldap_set_session_option()`

prldap_get_socket_info()

The `prldap_get_socket_info()` function retrieves socket-specific information based on parameters passed to the extended I/O callback functions.

Syntax

```
#include <ldappr.h>
int prldap_get_socket_info( int fd, void *socketarg,
    PRLDAPSocketInfo *soip );
```

Parameters

This function has the parameters listed in Table 17-219.

Table 17-219 prldap_get_socket_info() Function Parameters

Parameter	Description
fd	The session handle returned from prldap_init().
soip	Pointer to a structure containing socket information data.

Returns

- LDAP_SUCCESS if successful.
- An LDAP API error code if an error occurs.

See Also

prldap_set_socket_info()

prldap_init()

The prldap_init() function creates a new session handle with Netscape Portable Runtime (NSPR) I/O, threading, and DNS functions installed.

Syntax

```
#include <ldappr.h>
LDAP * prldap_init( const char *defhost, int defport, int shared );
```

Parameters

This function has the parameters listed in Table 17-220.

Table 17-220 prldap_init() Function Parameters

Parameter	Description
defhost	Space-delimited list of one or more host names (or IP address in dotted notation, such as "141.211.83.36") of the LDAP servers that you want the LDAP client to connect to. The names can be in <i>hostname:portnumber</i> format (in which case, <i>portnumber</i> overrides the port number specified by the defport argument.
defport	Default port number of the LDAP server. To specify the standard LDAP port (port 389), use LDAP_PORT as the value for this parameter.
shared	Pass a non-zero value if you plan to use this LDAP handle for more than one thread.

Returns

- Pointer to LDAP session handle if successful.
- NULL if an error occurs.

Description

To be able to use LDAP over Internet Protocol, version 6 (IPv6), this function should be used rather than `ldap_init()`.

See Also

`prldap_install_routines()`

prldap_install_routines()

The `prldap_install_routines()` function installs Netscape Portable Runtime (NSPR) I/O, threading, and DNS functions so they can be used by the LDAP session handle.

Syntax

```
#include <ldappr.h>
int prldap_install_routines( LDAP *ld, int shared );
```

Parameters

This function has the parameters listed in Table 17-221.

Table 17-221 `prldap_install_routines()` Function Parameters

Parameter	Description
ld	The session handle returned from <code>prldap_init()</code> . If NULL, the functions are installed as the default functions for all new LDAP handles.
shared	Pass a non-zero value if you plan to use this LDAP handle for more than one thread.

Returns

- LDAP_SUCCESS if successful.
- An LDAP API error code if an error occurs.

See Also

`prldap_init()`

prldap_set_default_socket_info()

The prldap_set_default_socket_info() function sets default socket information.

Syntax

```
#include <ldappr.h>
int prldap_set_default_socket_info( LDAP *ld, PRLDAPSocketInfo *soip );
```

Parameters

This function has the parameters listed in Table 17-225.

Table 17-222 prldap_set_default_socket_info() Function Parameters

Parameter	Description
fd	The session handle returned from prldap_init().
soip	Pointer to a structure containing socket specific information.

Returns

- LDAP_SUCCESS if successful.
- LDAP_PARAM_ERROR is returned if ld is NULL.
- An LDAP API error code if an error occurs.

See Also

prldap_get_socket_info()

prldap_set_session_info()

The prldap_set_session_info() function sets application-specific data.

Syntax

```
#include <ldappr.h>
int prldap_set_session_info( LDAP *ld, void *sessionarg,
    PRLDAPSessionInfo *seip );
```

Parameters

This function has the parameters listed in Table 17-223.

Table 17-223 prldap_set_session_info() Function Parameters

Parameter	Description
ld	The session handle returned from prldap_init().
seip	Pointer to a structure containing session information data.

Returns

- LDAP_SUCCESS if successful.
- An LDAP API error code if an error occurs.

Description

The prldap_set_session_info() function sets application-specific data for an LDAP session handle or a session argument that is passed to the CONNECT, POLL, NEWHANDLE, or DISPOSEHANDLE extended I/O callbacks.

See Also

prldap_get_session_info()

prldap_set_session_option()

The prldap_set_session_option() function sets a session option specific to the prldap layer.

Syntax

```
#include <ldappr.h>
int prldap_set_session_option( LDAP *ld, void *sessionarg,
    int option, ... );
```

Parameters

This function has the parameters listed in Table 17-224.

Table 17-224 prldap_set_session_option() Function Parameters

Parameter	Description
ld	The session handle returned from prldap_init().
option	

Returns

- LDAP_SUCCESS if successful.
- An LDAP API error code if an error occurs.

Description

The prldap_set_session_option() function sets an option for an LDAP session handle or a session argument that is passed to the CONNECT, POLL, NEWHANDLE, or DISPOSEHANDLE extended I/O callbacks.

See Also

prldap_get_session_option()

prldap_set_socket_info()

The prldap_set_socket_info() function sets socket information.

Syntax

```
#include <ldappr.h>
int prldap_set_socket_info( int fd, void *socketarg,
    PRLDAPSocketInfo *soip );
```

Parameters

This function has the parameters listed in Table 17-225.

Table 17-225 prldap_set_socket_info() Function Parameters

Parameter	Description
fd	The session handle returned from prldap_init().
soip	Pointer to a structure containing socket information data.

Returns

- LDAP_SUCCESS if successful.
- An LDAP API error code if an error occurs.

Description

The prldap_set_socket_info() function sets socket-specific information based on parameters passed to the extended I/O callback functions. For more information, see the ldappr.h header file.

See Also

prldap_get_socket_info()

`prldap_set_socket_info()`

Result Codes

This chapter lists some of the result codes that can be returned by functions in the Sun™ Java System LDAP SDK for C. For ease of use, they are first listed in numerical order, grouped by related occurrences. The detailed reference of each result code is then given in alphabetical order. This chapter contains the following sections:

- Overview
- Result Codes Summary in Numerical Order
- Result Codes Summary By Server / Client
- Result Codes Reference in Alphabetical Order

Overview

LDAP result codes are extensible; thus, LDAPv3 extensions may define their own error codes, and register them with the Internet Assigned Numbers Authority (IANA). The list of registered LDAP error codes is at <http://www.iana.org/assignments/ldap-parameters>. This list is what the LDAP SDK for C currently knows in terms of result codes. More information can be found in the *RFC 3383 - Internet Assigned Numbers Authority (IANA) Considerations for the Lightweight Directory Access Protocol (LDAP)* (<http://www.faqs.org/rfcs/rfc3383.html>).

Result Codes Summary in Numerical Order

Table 18-1 gives the decimal and hexadecimal value of all result codes. Values missing from the sequence are not assigned to a result code.

Table 18-1 Numerical Listing of Result Codes by Value

Decimal	Hexidecimal	Defined Name
0	0x00	LDAP_SUCCESS
1	0x01	LDAP_OPERATIONS_ERROR
2	0x02	LDAP_PROTOCOL_ERROR
3	0x03	LDAP_TIMELIMIT_EXCEEDED
4	0x04	LDAP_SIZELIMIT_EXCEEDED
5	0x05	LDAP_COMPARE_FALSE
6	0x06	LDAP_COMPARE_TRUE
7	0x07	LDAP_STRONG_AUTH_NOT_SUPPORTED
8	0x08	LDAP_STRONG_AUTH_REQUIRED
9	0x09	LDAP_PARTIAL_RESULTS
10	0x0a	LDAP_REFERRAL
11	0x0b	LDAP_ADMINLIMIT_EXCEEDED
12	0x0c	LDAP_UNAVAILABLE_CRITICAL_EXTENSION
13	0x0d	LDAP_CONFIDENTIALITY_REQUIRED
14	0x0e	LDAP_SASL_BIND_IN_PROGRESS
16	0x10	LDAP_NO_SUCH_ATTRIBUTE
17	0x11	LDAP_UNDEFINED_TYPE
18	0x12	LDAP_INAPPROPRIATE_MATCHING
19	0x13	LDAP_CONSTRAINT_VIOLATION
20	0x14	LDAP_TYPE_OR_VALUE_EXISTS
21	0x15	LDAP_INVALID_SYNTAX
32	0x20	LDAP_NO_SUCH_OBJECT
33	0x21	LDAP_ALIAS_PROBLEM
34	0x22	LDAP_INVALID_DN_SYNTAX
35	0x23	LDAP_IS_LEAF
36	0x24	LDAP_ALIAS_DEREF_PROBLEM

Table 18-1 Numerical Listing of Result Codes by Value (*Continued*)

Decimal	Hexidecimal	Defined Name
48	0x30	LDAP_INAPPROPRIATE_AUTH
49	0x31	LDAP_INVALID_CREDENTIALS
50	0x32	LDAP_INSUFFICIENT_ACCESS
51	0x33	LDAP_BUSY
52	0x34	LDAP_UNAVAILABLE
53	0x35	LDAP_UNWILLING_TO_PERFORM
54	0x36	LDAP_LOOP_DETECT
60	0x3C	LDAP_SORT_CONTROL_MISSING
61	0x3D	LDAP_INDEX_RANGE_ERROR
64	0x40	LDAP_NAMING_VIOLATION
65	0x41	LDAP_OBJECT_CLASS_VIOLATION
66	0x42	LDAP_NOT_ALLOWED_ON_NONLEAF
67	0x43	LDAP_NOT_ALLOWED_ON_RDN
68	0x44	LDAP_ALREADY_EXISTS
69	0x45	LDAP_NO_OBJECT_CLASS_MODS
70	0x46	LDAP_RESULTS_TOO_LARGE
71	0x47	LDAP_AFFECTS_MULTIPLE_DSAS
80	0x50	LDAP_OTHER
81	0x51	LDAP_SERVER_DOWN
82	0x52	LDAP_LOCAL_ERROR
83	0x53	LDAP_ENCODING_ERROR
84	0x54	LDAP_DECODING_ERROR
85	0x55	LDAP_TIMEOUT
86	0x56	LDAP_AUTH_UNKNOWN
87	0x57	LDAP_FILTER_ERROR
88	0x58	LDAP_USER_CANCELLED
89	0x59	LDAP_PARAM_ERROR

Table 18-1 Numerical Listing of Result Codes by Value *(Continued)*

Decimal	Hexidecimal	Defined Name
90	0x5a	LDAP_NO_MEMORY
91	0x5b	LDAP_CONNECT_ERROR
92	0x5c	LDAP_NOT_SUPPORTED
93	0x5d	LDAP_CONTROL_NOT_FOUND
94	0x5e	LDAP_NO_RESULTS_RETURNED
95	0x5f	LDAP_MORE_RESULTS_TO_RETURN
96	0x60	LDAP_CLIENT_LOOP
97	0x61	LDAP_REFERRAL_LIMIT_EXCEEDED

Result Codes Summary By Server/Client

Table 18-2 summarizes the result codes initiated by an LDAP server.

Table 18-2 LDAP Server Result Codes

LDAP_ADMINLIMIT_EXCEEDED	LDAP_NOT_ALLOWED_ON_NONLEAF
LDAP_AFFECTS_MULTIPLE_DSAS	LDAP_NOT_ALLOWED_ON_RDN
LDAP_ALIAS_DEREF_PROBLEM	LDAP_OBJECT_CLASS_VIOLATION
LDAP_ALIAS_PROBLEM	LDAP_OPERATIONS_ERROR
LDAP_ALREADY_EXISTS	LDAP_OTHER
LDAP_BUSY	LDAP_PARTIAL_RESULTS
LDAP_COMPARE_FALSE	LDAP_PROTOCOL_ERROR
LDAP_COMPARE_TRUE	LDAP_REFERRAL
LDAP_CONFIDENTIALITY_REQUIRED	LDAP_RESULTS_TOO_LARGE
LDAP_CONSTRAINT_VIOLATION	LDAP_SASL_BIND_IN_PROGRESS
LDAP_INAPPROPRIATE_AUTH	LDAP_SIZELIMIT_EXCEEDED
LDAP_INAPPROPRIATE_MATCHING	LDAP_SORT_CONTROL_MISSING
LDAP_INSUFFICIENT_ACCESS	LDAP_STRONG_AUTH_NOT_SUPPORTED
LDAP_INVALID_CREDENTIALS	LDAP_STRONG_AUTH_REQUIRED
LDAP_INVALID_DN_SYNTAX	LDAP_TIMELIMIT_EXCEEDED

Table 18-2 LDAP Server Result Codes (*Continued*)

LDAP_INVALID_SYNTAX	LDAP_TYPE_OR_VALUE_EXISTS
LDAP_IS_LEAF	LDAP_UNAVAILABLE
LDAP_LOOP_DETECT	LDAP_UNAVAILABLE_CRITICAL_EXTENSION
LDAP_NAMING_VIOLATION	LDAP_UNDEFINED_TYPE
LDAP_NO_OBJECT_CLASS_MODS	LDAP_UNWILLING_TO_PERFORM
LDAP_NO_SUCH_ATTRIBUTE	LDAP_USER_CANCELLED
LDAP_NO_SUCH_OBJECT	

Table 18-3 summarizes the result codes initiated by an LDAP client.

Table 18-3 LDAP Client Result Codes

LDAP_AUTH_UNKNOWN	LDAP_MORE_RESULTS_TO_RETURN
LDAP_CLIENT_LOOP	LDAP_NO_MEMORY
LDAP_CONNECT_ERROR	LDAP_NO_RESULTS_RETURNED
LDAP_CONTROL_NOT_FOUND	LDAP_NOT_SUPPORTED
LDAP_DECODING_ERROR	LDAP_PARAM_ERROR
LDAP_ENCODING_ERROR	LDAP_REFERRAL_LIMIT_EXCEEDED
LDAP_FILTER_ERROR	LDAP_SERVER_DOWN
LDAP_INDEX_RANGE_ERROR	LDAP_SUCCESS
LDAP_LOCAL_ERROR	LDAP_TIMEOUT

Result Codes Reference in Alphabetical Order

The following sections contain the detailed reference information for each result code listed in alphabetical order by code name.

LDAP_ADMINLIMIT_EXCEEDED

This result code indicates that the look-through limit on a search operation has been exceeded. The *look-through limit* is the maximum number of entries that the server will check when gathering a list of potential search result candidates. For more information, see the *Sun ONE Directory Server Administration Guide* (<http://docs.sun.com/doc/816-6698-10>).

NOTE When working with Sun Java System Directory Server, keep in mind the following:

- If you are bound as the root DN, the server sets an infinite look-through limit.
- If you are not bound as the root DN, the server sets a time limit to the value specified by the `lookthroughtime limit` directive in the server's `slapd.conf` configuration file.

Definition

```
#define LDAP_ADMINLIMIT_EXCEEDED 0x0b /* 11 */
```

LDAP_AFFECTS_MULTIPLE_DSAS

This result code indicates that the requested operation needs to be performed on multiple servers, where this operation is not permitted.

Definition

```
#define LDAP_AFFECTS_MULTIPLE_DSAS 0x47 /* 71 */
```

LDAP_ALIAS_DEREF_PROBLEM

This result code indicates that a problem occurred when dereferencing an alias.

CAUTION Sun Java System Directory Server does not currently send this result code back to LDAP clients.

Definition

```
#define LDAP_ALIAS_DEREF_PROBLEM 0x24 /* 36 */
```

LDAP_ALIAS_PROBLEM

This result code indicates that the alias is invalid.

CAUTION Sun Java System Directory Server does not currently send this result code back to LDAP clients.

Definition

```
#define LDAP_ALIAS_PROBLEM 0x21 /* 33 */
```


LDAP_ALREADY_EXISTS

This result code indicates that the request is attempting to add an entry that already exists in the directory. Sun Java System Directory Server sends this result code back to the client in the following situations:

- The request is an add request, and the entry already exists in the directory.
- The request is a modify DN request, and the new DN of the entry already identifies another entry.
- The request is adding an attribute to the schema, and an attribute with the specified name or object identifier (OID) already exists.

Definition

```
#define LDAP_ALREADY_EXISTS 0x44 /* 68 */
```

LDAP_AUTH_UNKNOWN

This result code indicates that an unknown authentication method was specified.

NOTE

The LDAP SDK for C library sets this result code if `ldap_bind()` or `ldap_bind_s()` are called and an authentication method other than `LDAP_AUTH_SIMPLE` is specified. (These functions only allow you to use simple authentication.)

Definition

```
#define LDAP_AUTH_UNKNOWN 0x56 /* 86 */
```

LDAP_BUSY

This result code indicates that the server is currently too busy to perform the requested operation.

Definition

```
#define LDAP_BUSY 0x33 /* 51 */
```

LDAP_CLIENT_LOOP

This result code indicates that the LDAP client detected a loop, for example, when following referrals.

Definition

```
#define LDAP_CLIENT_LOOP 0x60 /* 96 */
```

LDAP_COMPARE_FALSE

This result code is returned after an LDAP compare operation is completed. The result indicates that the specified attribute value is not present in the specified entry.

Definition

```
#define LDAP_COMPARE_FALSE 0x05 /* 5 */
```

LDAP_COMPARE_TRUE

This result code is returned after an LDAP compare operation is completed. The result indicates that the specified attribute value is present in the specified entry.

Definition

```
#define LDAP_COMPARE_TRUE 0x06 /* 6 */
```

LDAP_CONFIDENTIALITY_REQUIRED

This result code indicates that confidentiality is required for the operation.

Definition

```
#define LDAP_CONFIDENTIALITY_REQUIRED 0x0d /* 13 */
```

LDAP_CONNECT_ERROR

This result code indicates that the LDAP client cannot establish a connection, or has lost the connection, with the LDAP server. The LDAP SDK for C sets this result code. If you have not established an initial connection with the server, verify that you have specified the correct host name and port number and that the server is running. If you have lost the connection to the server, see “Handling Failover” on page 100 of Chapter 4, “Using the LDAP API” for instructions on reconnecting.

Definition

```
#define LDAP_CONNECT_ERROR 0x5b /* 91 */
```

LDAP_CONSTRAINT_VIOLATION

This result code indicates that a value in the request does not comply with certain constraints. Sun Java System Directory Server sends this result code back to the client in the following situations:

- The request adds or modifies the `userpassword` attribute, and one of the following is true:

- The server is configured to check the password syntax, and the length of the new password is less than the minimum password length.
- The server is configured to check the password syntax, and the new password is the same as one of the values of the `uid`, `cn`, `sn`, `givenname`, `ou`, or `mail` attributes.
- The server is configured to keep a history of previous passwords, and the new password is the same as one of the previous passwords.
- The request is a bind request, and the user is locked out of the account. (For example, the server can be configured to lock a user out of the account after a given number of failed attempts to bind to the server.)

Definition

```
#define LDAP_CONSTRAINT_VIOLATION 0x13 /* 19 */
```

LDAP_CONTROL_NOT_FOUND

This result code indicates that a requested LDAP control was not found. The LDAP SDK for C sets this result code when parsing a server response for controls and not finding the requested controls. For example:

- `ldap_parse_entrychange_control()` is called, but no entry change notification control is found in the server's response.
- `ldap_parse_sort_control()` is called, but no server-side sorting control is found in the server's response.
- `ldap_parse_virtuallist_control()` is called, but no virtual list view response control is found in the server's response.

For more information on controls, see Chapter 13, “Working with LDAP Controls.”

Definition

```
#define LDAP_CONTROL_NOT_FOUND 0x5d /* 93 */
```

LDAP_DECODING_ERROR

This result code indicates that the LDAP client encountered an error when decoding the LDAP response received from the server.

Definition

```
#define LDAP_DECODING_ERROR 0x54 /* 84 */
```

LDAP_ENCODING_ERROR

This result code indicates that the LDAP client encountered an error when encoding the LDAP request to be sent to the server.

Definition

```
#define LDAP_ENCODING_ERROR 0x53 /* 83 */
```

LDAP_FILTER_ERROR

This result code indicates that an error occurred when specifying the search filter. The LDAP SDK for C sets this result code if it cannot encode the specified search filter in an LDAP search request.

Definition

```
#define LDAP_FILTER_ERROR 0x57 /* 87 */
```

LDAP_INAPPROPRIATE_AUTH

This result code indicates that the type of credentials are not appropriate for the method of authentication used. Sun Java System Directory Server sends this result code back to the client if simple authentication is used in a bind request, but the entry has no `userpassword` attribute; also, if `LDAP_SASL_EXTERNAL` is attempted on a non-SSL connection.

Definition

```
#define LDAP_INAPPROPRIATE_AUTH 0x30 /* 48 */
```

LDAP_INAPPROPRIATE_MATCHING

This result code indicates that an extensible match filter in a search request contained a matching rule that does not apply to the specified attribute type.

Definition

```
#define LDAP_INAPPROPRIATE_MATCHING 0x12 /* 18 */
```

LDAP_INDEX_RANGE_ERROR

This result code indicates that the search results exceeded the range specified by the requested offsets. This result code applies to search requests that contain *virtual list view* controls. For more information on this control, see “Using the Virtual List View Control” on page 245 of Chapter 13, “Working with LDAP Controls.”

NOTE

Versions of Directory Server prior to 4.0 do not support the virtual list view control.

Definition

```
#define LDAP_INDEX_RANGE_ERROR 0x3D /* 61 */
```

LDAP_INSUFFICIENT_ACCESS

This result code indicates that the client has insufficient access to perform the operation. Check that the user you are authenticating as has the appropriate permissions. For information on managing the access control, see the *Sun ONE Directory Server Administration Guide* (<http://docs.sun.com/doc/816-6698-10>).

Definition

```
#define LDAP_INSUFFICIENT_ACCESS 0x32 /* 50 */
```

LDAP_INVALID_CREDENTIALS

This result code indicates that the credentials provided in the request are invalid. Sun Java System Directory Server sends this result code back to the client if a bind request contains the incorrect credentials for a user or if a user's password has already expired.

Definition

```
#define LDAP_INVALID_CREDENTIALS 0x31 /* 49 */
```

LDAP_INVALID_DN_SYNTAX

This result code indicates that an invalid DN has been specified. Sun Java System Directory Server sends this result code back to the client if an add request or a modify DN request specifies an invalid DN. It also sends this code when an LDAP_SASL_EXTERNAL bind is attempted but certification to DN mapping fails.

Definition

```
#define LDAP_INVALID_DN_SYNTAX 0x22 /* 34 */
```

LDAP_INVALID_SYNTAX

This result code indicates that the request contains invalid syntax. Sun Java System Directory Server sends this result code back to the client in the following situations:

- The server encounters an access control instruction (ACI) with invalid syntax.
- The request attempts to add or modify an aci attribute, and the value of the attribute is an ACI with invalid syntax.
- The request is a search request with a substring filter, and the syntax of the filter is invalid.

- The request is a modify request that is attempting to modify the schema, but no values are provided (for example, the request might be attempting to delete all values of the objectclass attribute).

Definition

```
#define LDAP_INVALID_SYNTAX 0x15 /* 21 */
```

LDAP_IS_LEAF

This result code indicates that the specified entry is a leaf entry.

CAUTION Sun Java System Directory Server does not currently send this result code back to LDAP clients.

Definition

```
#define LDAP_IS_LEAF 0x23 /* 35 */
```

LDAP_LOCAL_ERROR

This result code indicates that an error occurred in the LDAP client.

Definition

```
#define LDAP_LOCAL_ERROR 0x52 /* 82 */
```

LDAP_LOOP_DETECT

This result code indicates that the server was unable to perform the requested operation because of an internal loop.

CAUTION Sun Java System Directory Server does not currently send this result code back to LDAP clients.

Definition

```
#define LDAP_LOOP_DETECT 0x36 /* 54 */
```

LDAP_MORE_RESULTS_TO_RETURN

This result code indicates that there are more results in the chain of results. The LDAP SDK for C sets this result code when the `ldap_parse_sasl_bind_result()` function is called to retrieve the result code of an operation, and additional result codes from the server are available in the LDAP structure.

Definition

```
#define LDAP_MORE_RESULTS_TO_RETURN 0x5f /* 95 */
```

LDAP_NAMING_VIOLATION

This result code indicates that the request violates the structure of the DIT.

CAUTION Sun Java System Directory Server does not currently send this result code back to LDAP clients.

Definition

```
#define LDAP_NAMING_VIOLATION 0x40 /* 64 */
```

LDAP_NO_MEMORY

This result code indicates that no memory is available. The LDAP SDK for C sets this result code if a function cannot allocate memory (for example, when creating an LDAP request or an LDAP control).

Definition

```
#define LDAP_NO_MEMORY 0x5a /* 90 */
```

LDAP_NO_OBJECT_CLASS_MODS

This result code indicates that the request is attempting to modify an object class that should not be modified (for example, a structural object class).

CAUTION Sun Java System Directory Server does not currently send this result code back to LDAP clients.

Definition

```
#define LDAP_NO_OBJECT_CLASS_MODS 0x45 /* 69 */
```

LDAP_NO_RESULTS_RETURNED

This result code indicates that no results were returned from the server. The LDAP SDK for C sets this result code when the `ldap_parse_result()` function is called but no result code is included in the server's response.

Definition

```
#define LDAP_NO_RESULTS_RETURNED 0x5E /* 94 */
```

LDAP_NO_SUCH_ATTRIBUTE

This result code indicates that the specified attribute does not exist in the entry. Sun Java System Directory Server might send this result code back to the client if, for example, a modify request specifies the modification or removal of a non-existent attribute or if a compare request specifies a non-existent attribute.

Definition

```
#define LDAP_NO_SUCH_ATTRIBUTE 0x10 /* 16 */
```

LDAP_NO_SUCH_OBJECT

This result code indicates that the server cannot find an entry specified in the request. Sun Java System Directory Server sends this result code back to the client if it cannot find a requested entry and it cannot refer your client to another LDAP server.

Definition

```
#define LDAP_NO_SUCH_OBJECT 0x20 /* 32 */
```

LDAP_NOT_ALLOWED_ON_NONLEAF

This result code indicates that the requested operation is allowed only on entries that do not have child entries (*leaf* entries as opposed to *branch* entries). Sun Java System Directory Server sends this result code back to the client if the request is a delete request or a modify DN request and the entry is a parent entry. (You cannot delete or move a branch of entries in a single operation.)

Definition

```
#define LDAP_NOT_ALLOWED_ON_NONLEAF 0x42 /* 66 */
```

LDAP_NOT_ALLOWED_ON_RDN

This result code indicates that the requested operation will affect the RDN of the entry. Sun Java System Directory Server sends this result code back to the client if the request is a modify request that deletes attribute values from the entry that are used in the RDN of the entry. (For example, the request removes the attribute value uid=bjensen from the entry uid=bjensen,ou=People,dc=example,dc=com.)

Definition

```
#define LDAP_NOT_ALLOWED_ON_RDN 0x43 /* 67 */
```


LDAP_NOT_SUPPORTED

This result code indicates that the LDAP client is attempting to use functionality that is not supported. The LDAP SDK for C sets this result code if the client identifies itself as an LDAPv2 client, and the client is attempting to use functionality available in LDAPv3. For example:

- You are passing LDAP controls to a function.
- You are calling `ldap_extended_operation()`, `ldap_extended_operation_s()`, or `ldap_parse_extended_result()` to request an extended operation or to parse an extended response.
- You are calling `ldap_rename()` or `ldap_rename_s()`, and you are specifying a new "superior DN" as an argument.
- You are calling `ldap_sasl_bind()`, `ldap_sasl_bind_s()`, or `ldap_parse_sasl_bind_result()` to request Simple Authentication and Security Layer (SASL) authentication or to parse a SASL bind response.
- You are calling `ldap_parse_virtuallist_control()` to parse a virtual list control from the server's response.

If you want to use these features, make sure to specify that your LDAP client is an LDAPv3 client. For details, see "Specifying the LDAP Version of Your Client" on page 65 of Chapter 3, "Writing an LDAP Client."

Definition

```
#define LDAP_NOT_SUPPORTED 0x5c /* 92 */
```

LDAP_OBJECT_CLASS_VIOLATION

This result code indicates that the request specifies a new entry or a change to an existing entry that does not comply with the server's schema. Sun Java System Directory Server sends this result code back to the client in the following situations:

- The request is an add request, and the new entry does not comply with the schema. For example, the new entry does not have all the required attributes, or the entry has attributes that are not allowed in the entry.
- The request is a modify request, and the change will make the entry non-compliant with the schema. For example, the change removes a required attribute or adds an attribute that is not allowed.

Check the server error logs for more information, and the schema for the type of entry that you are adding or modifying. For information on the schema, see the *Sun ONE Directory Server Administration Guide* (<http://docs.sun.com/doc/816-6698-10>).

Definition

```
#define LDAP_OBJECT_CLASS_VIOLATION 0x41 /* 65 */
```

LDAP_OPERATIONS_ERROR

This is a general result code indicating that an error has occurred. Sun Java System Directory Server might send this code if, for example, memory cannot be allocated on the server. To troubleshoot this type of error, check the server's error logs. You may need to increase the log level of the server to get additional information. For information on setting the log level, see the *Sun ONE Directory Server Administration Guide* (<http://docs.sun.com/doc/816-6698-10>).

Definition

```
#define LDAP_OPERATIONS_ERROR 0x01 /* 1 */
```

LDAP_OTHER

This result code indicates than an unknown error has occurred.

CAUTION At this point, neither the LDAP SDK for C nor Sun Java System Directory Server return this result code.

Definition

```
#define LDAP_OTHER 0x50 /* 80 */
```

LDAP_PARAM_ERROR

This result code indicates that an invalid parameter was specified. The LDAP SDK for C sets this result code if a function was called and invalid parameters were specified (for example, if the LDAP structure is NULL).

Definition

```
#define LDAP_PARAM_ERROR 0x59 /* 89 */
```

LDAP_PARTIAL_RESULTS

Sun Java System Directory Server sends this result code to LDAPv2 clients to refer them to another LDAP server. When sending this code to a client, the server includes a new line-delimited list of LDAP URLs that identifies another LDAP server. If the client identifies itself as an LDAPv3 client in the request, an LDAP_REFERRAL result code is sent instead of this result code.

Definition

```
#define LDAP_PARTIAL_RESULTS 0x09 /* 9 */
```

LDAP_PROTOCOL_ERROR

This result code indicates that the LDAP client's request does not comply with the LDAP. Sun Java System Directory Server sends this result code back to the client in the following situations:

- The server cannot parse the incoming request.
- The request specifies an attribute type that uses a syntax not supported by the server.
- The request is a SASL bind request, but your client identifies itself as an LDAPv2 client. (For instructions on how to specify that your LDAP client is an LDAPv3 client, see “Specifying the LDAP Version of Your Client” on page 65 of Chapter 3, “Writing an LDAP Client.”)
- The request is a bind request that specifies an unsupported version of the LDAP. Make sure to specify whether your LDAP client is an LDAPv2 client or an LDAPv3 client. For instructions, see “Specifying the LDAP Version of Your Client” on page 65 of Chapter 3, “Writing an LDAP Client.”
- The request is an add or a modify request that specifies the addition of an attribute type to an entry, but no values are specified.
- The request is a modify request, and one of the following is true:
 - An unknown modify operation is specified (an operation other than LDAP_MOD_ADD, LDAP_MOD_DELETE, and LDAP_MOD_REPLACE).
 - No modifications are specified.
- The request is a modify DN request, and one of the following is true:
 - The new RDN is not a valid RDN.
 - A new superior DN is specified, but your client identifies itself as an LDAPv2 client.
- The request is a search request, and one of the following is true:
 - An unknown scope is specified (a scope other than LDAP_SCOPE_BASE, LDAP_SCOPE_ONELEVEL, and LDAP_SCOPE_SUBTREE).
 - An unknown filter type is specified.

- The filter type `LDAP_FILTER_GE` or `LDAP_FILTER_LE` is specified, but the type of attribute contains values that cannot be ordered. (For example, if the attribute type uses a binary syntax, the values of the attribute contain binary data, which cannot be sorted.)
 - The request contains an extensible filter (a filter using matching rules), but your client identifies itself as an LDAPv2 client. (For instructions on how to specify that your LDAP client is an LDAPv3 client, see “Specifying the LDAP Version of Your Client” on page 65 of Chapter 3, “Writing an LDAP Client.”)
 - The request contains an extensible filter (a filter using matching rules), but the matching rule is not supported by the server.
- The request is a search request with a server-side sorting control, and one of the following is true:
 - The server does not have a syntax plug-in that supports the attribute used for sorting.
 - The syntax plug-in does not have a function for comparing values of the attribute. (This compare function is used for sorting.)
 - The type of attribute specified for sorting contains values that cannot be sorted in any order. For example, if the attribute type uses a binary syntax, the values of the attribute contain binary data, which cannot be sorted.
 - The server encounters an error when creating the sorting response control (the control to be sent back to the client).
 - When sorting the results, the time limit or the "look-through limit" is exceeded. The "look-through limit" is the maximum number of entries that the server will check when gathering a list of potential search result candidates. For details, see the *Sun ONE Directory Server Administration Guide* (<http://docs.sun.com/doc/816-6698-10>).
- The request is an extended operation request, and the server does not support the extended operation. In Directory Server, extended operations are supported through extended operation server plug-ins. Make sure that the server is loading a plug-in that supports the extended operation. Check the OID of the extended operation in your LDAP client to make sure that it matches the OID of the extended operation registered in the server plug-in. For more information on extended operation server plug-ins, see the *Sun ONE Directory Server Plug-In Programming Guide* (<http://docs.sun.com/doc/816-6702-10>).

- An authentication method other than LDAP_AUTH_SIMPLE or LDAP_AUTH_SASL is specified.

To troubleshoot this type of error, check the server's error logs. You may need to increase the log level of the server to get additional information. See the *Sun ONE Directory Server Administration Guide* (<http://docs.sun.com/doc/816-6698-10>) for information on setting the log level of the server.

Definition

```
#define LDAP_PROTOCOL_ERROR 0x02 /* 2 */
```

LDAP_REFERRAL

This result code indicates that the server is referring the client to another LDAP server. When sending this code to a client, the server includes a list of LDAP URLs that identify another LDAP server. This result code is part of the LDAPv3. For LDAPv2 clients, Sun Java System Directory Server sends an LDAP_PARTIAL_RESULTS result code instead.

Definition

```
#define LDAP_REFERRAL 0x0a /* 10 */
```

LDAP_REFERRAL_LIMIT_EXCEEDED

This result code indicates that the *referral hop limit* was exceeded. The LDAP SDK for C sets this result code, when following referrals, if the client is referred to other servers more times than allowed by the *referral hop limit*. For more information, see “Limiting Referral Hops” on page 95 of Chapter 4, “Using the LDAP API.”

Definition

```
#define LDAP_REFERRAL_LIMIT_EXCEEDED 0x61 /* 97 */
```

LDAP_RESULTS_TOO_LARGE

This result code indicates that the results of the request are too large.

CAUTION Sun Java System Directory Server does not currently send this result code back to LDAP clients.

Definition

```
#define LDAP_RESULTS_TOO_LARGE 0x46 /* 70 */
```

LDAP_SASL_BIND_IN_PROGRESS

This result code is used in multi-stage SASL bind operations. The server sends this result code back to the client to indicate that the authentication process has not yet completed. For more information on SASL authentication, see Chapter 12, “Using SASL Authentication.”

Definition

```
#define LDAP_SASL_BIND_IN_PROGRESS 0x0E /* 14 */
```

LDAP_SERVER_DOWN

This result code indicates that the LDAP SDK for C cannot establish a connection with, or lost the connection to, the LDAP server. If you have not established an initial connection with the server, verify that you have specified the correct host name and port number and that the server is running. If you have lost the connection to the server, see “Handling Failover” on page 100 of Chapter 4, “Using the LDAP API” for instructions on reconnecting to the server.

Definition

```
#define LDAP_SERVER_DOWN 0x51 /* 81 */
```

LDAP_SIZELIMIT_EXCEEDED

This result code indicates that the maximum number of search results to return has been exceeded. This limit is specified in the search request. If you specify no size limit, the server will set one. When working with Sun Java System Directory Server, keep in mind the following:

- If you are bound as the root DN and specify no size limit, the server enforces no size limit at all.
- If you are not bound as the root DN and specify no size limit, the server sets the size limit to the value specified by the `sizelimit` directive in the server’s `slapd.conf` configuration file.
- If the size limit that you specify exceeds the value specified by the `sizelimit` directive in the server’s `slapd.conf` configuration file, the server uses the size limit specified in the configuration file.

Definition

```
#define LDAP_SIZELIMIT_EXCEEDED 0x04 /* 4 */
```

LDAP_SORT_CONTROL_MISSING

This result code indicates that server did not receive a required server-side sorting control. Directory Server 4.0 or later sends this result code back to the client if the server receives a search request with a *virtual list view* control but no server-side sorting control as the *virtual list view* control requires a server-side sorting control. For more information on this, see “Using the Virtual List View Control” on page 245 of Chapter 13, “Working with LDAP Controls.”

NOTE Versions of Directory Server prior to 4.0 do not support the virtual list view control.

Definition

```
#define LDAP_SORT_CONTROL_MISSING 0x3C /* 60 */
```

LDAP_STRONG_AUTH_NOT_SUPPORTED

This result code is returned as the result of a bind operation. It indicates that the server does not recognize or support the specified authentication method.

Definition

```
#define LDAP_STRONG_AUTH_NOT_SUPPORTED 0x07 /* 7 */
```

LDAP_STRONG_AUTH_REQUIRED

This result code indicates that a stronger method of authentication is required to perform the operation.

Definition

```
#define LDAP_STRONG_AUTH_REQUIRED 0x08 /* 8 */
```

LDAP_SUCCESS

This result code indicates that the LDAP operation was successful.

Definition

```
#define LDAP_SUCCESS 0x00 /* 0 */
```

LDAP_TIMELIMIT_EXCEEDED

This result code indicates that the time limit on a search operation has been exceeded. The time limit is specified in the search request. If you specify no time limit, the server will set one. When working with Sun Java System Directory Server, keep in mind the following:

- If you are bound as the root DN and specify no time limit, the server enforces no limit at all.
- If you are not bound as the root DN and specify no time limit, the server sets the time limit to the value specified by the `timelimit` directive in the server's `slapd.conf` configuration file.
- If the time limit that you specify exceeds the value specified by the `timelimit` directive in the server's `slapd.conf` configuration file, the server uses the time limit specified in the configuration file.

Definition

```
#define LDAP_TIMELIMIT_EXCEEDED 0x03 /* 3 */
```

LDAP_TIMEOUT

This result code indicates that the LDAP client timed out while waiting for a response from the server. The LDAP SDK for C sets this result code in the LDAP structure if the timeout period (for example, in a search request) has been exceeded and the server has not responded.

Definition

```
#define LDAP_TIMEOUT 0x55 /* 85 */
```

LDAP_TYPE_OR_VALUE_EXISTS

This result code indicates that the request attempted to add an attribute type or value that already exists. Sun Java System Directory Server sends this result code back to the client in the following situations:

- The request attempts to add values that already exist in the attribute.
- The request is adding an attribute to the schema of the server, but the OID of the attribute is already used by an object class in the schema.
- The request is adding an object class to the schema of the server, and one of the following occurs:
 - The object class already exists.
 - The OID of the object class is already used by another object class or an attribute in the schema.
 - The superior object class for this new object class does not exist.

Definition

```
#define LDAP_TYPE_OR_VALUE_EXISTS 0x14 /* 20 */
```


LDAP_UNAVAILABLE

This result code indicates that the server is unavailable to perform the requested operation.

CAUTION At this point, neither the LDAP SDK for C nor Sun Java System Directory Server return this result code.

Definition

```
#define LDAP_UNAVAILABLE 0x34 /* 52 */
```

LDAP_UNAVAILABLE_CRITICAL_EXTENSION

This result code indicates that the specified control or matching rule is not supported by the server. Sun Java System Directory Server might send back this result code if the request includes an unsupported control or if the filter in the search request specifies an unsupported matching rule.

Definition

```
#define LDAP_UNAVAILABLE_CRITICAL_EXTENSION 0x0c /* 12 */
```

LDAP_UNDEFINED_TYPE

This result code indicates that the request specifies an undefined attribute type.

CAUTION Sun Java System Directory Server does not currently send this result code back to LDAP clients.

Definition

```
#define LDAP_UNDEFINED_TYPE 0x11 /* 17 */
```

LDAP_UNWILLING_TO_PERFORM

This result code indicates that the server is unwilling to perform the requested operation. Sun Java System Directory Server sends this result code back to the client in the following situations:

- The client has logged in for the first time and needs to change its password, but the client is requesting to perform other LDAP operations. In this situation, the result code is accompanied by an expired password control. For details, see “Using Password Policy Controls” on page 245 of Chapter 13, “Working with LDAP Controls.”

- The NT Synch Service is running, and an operation is *vetoed* by the service.
- The request is a modify DN request, and a "superior DN" is specified. (At this point in time, Directory Server does not support the ability to use the modify DN operation to move an entry from one location in the directory tree to another location.)
- The database is in read-only mode, and the request attempts to write to the directory.
- The request is a delete request that attempts to delete the root DSE.
- The request is a modify DN request that attempts to modify the DN of the root DSE.
- The request is a modify request to modify the schema entry, and one of the following occurs:
 - The operation is LDAP_MOD_REPLACE. (The server does not allow you to replace schema entry attributes.)
 - The request attempts to delete an object class that is the parent of another object class.
 - The request attempts to delete a read-only object class or attribute.
- The server uses a database plug-in that does not implement the operation specified in the request. For example, if the database plug-in does not implement the add operation, sending an add request will return this result code.

Definition

```
#define LDAP_UNWILLING_TO_PERFORM 0x35 /* 53 */
```

LDAP_USER_CANCELLED

This result code indicates that the user cancelled the LDAP operation.

CAUTION Sun Java System Directory Server does not currently send this result code back to LDAP clients.

Definition

```
#define LDAP_USER_CANCELLED 0x58 /* 88 */
```

Glossary

base DN The distinguished name (DN) that identifies the starting point of a search. For example, if you want to search all entries under the `ou=People,dc=example,dc=com` subtree of the directory, `ou=People,dc=example,dc=com` is the base DN. For more information, see “Specifying the Base DN and Scope” on page 108 of Chapter 5, “Searching a LDAP Directory.”

client The calling module in the client/server architecture, a client is the requestor of services.

connection handle A pointer to the LDAP structure that contains information about the connection to the LDAP server.

control LDAP controls are specified as part of the LDAPv3. A control provides the means to specify additional information for an operation. Clients and servers can send controls as part of the requests and responses for an operation. For more information, see Chapter 13, “Working with LDAP Controls.”

DIT The hierarchical organization of entries that make up a directory. DIT stands for “Directory Information Tree.”

DSA An X.500 term for a directory server. DSA stands for “Directory System Agent.”

DSE An entry containing server-specific information. DSE stands for “DSA-specific entry.” Each server has different attribute values for the DSE. For more information, see “Understanding DSEs” on page 211 of Chapter 10, “Retrieving Server Information.”

extended operation An extension mechanism in the LDAPv3. You can define extended operations to perform services not covered by the protocol. The extended operation mechanism specifies the means for an LDAP client to request a custom operation (not specified in the LDAP) from an LDAP server. For more information, see Chapter 15, “Extended Operations.”

operational attributes Attributes that are used by servers for administering the directory. For example, `creatorsName` is an operational attribute that specifies the DN of the user who added the entry. Operational attributes are not returned in any search results unless you specify the attribute by name in the search request.

referral Refers an LDAP client to a different LDAP server. An LDAP server can be configured to send your client a referral if your client requests a DN with a suffix that is not in the server’s DIT. For example, if the directory includes entries under `dc=example,dc=com` and your client requests an entry under `dc=examplewest,dc=com`, the request would be referred to a different LDAP server. (Referrals contain LDAP URLs that specify the host, port, and base DN of another LDAP server.)

NOTE Although similar, referrals are not the same as search references. A search reference is returned as part of the results of a search; a referral is returned when the base DN of a search (or the target DN of any other LDAP operation) is not part of the LDAP server’s directory tree.

referral hop limit The maximum number of referrals that your client can follow in a row. For example, suppose your client receives a referral from LDAP server A to LDAP server B. After your client follows the referral to LDAP server B, that server sends you a referral to LDAP server C, which in turn refers you to LDAP server D. Your client has been referred 3 times in a row. If the referral hop limit is 2, the referral hop limit has been exceeded.

root DSE A DSE that is located at the root of the DIT. For more information, see “Getting the Root DSE” on page 212 of Chapter 10, “Retrieving Server Information.”

search reference Also known as continuation references, search result references, or smart referrals. A search reference is an entry in the directory that refers to another LDAP server (the reference is in the form of an LDAP URL). Search references are returned in search results along with entries found in the search. (A referral, on the other hand, is returned before searching through any entries. A referral is returned if the base DN does not have a suffix that is handled by the server.)

search result reference See search reference.

server The module from which services are requested in the client/server architecture.

server plug-in The Sun Java System Directory Server supports a plug-in interface that allows you to extend the functionality of the server. You can write plug-ins that handle extended operations or SASL authentication requests. For more information on server plug-ins, see the Directory Server 5.2 API Programming Guide or API Reference both located at http://docs.sun.com/coll/S1_DirectoryServer_52.

smart referral See search reference.

subschema entry Entry containing all the schema definitions (definitions of object classes, attributes, matching rules, and so on) used by entries in part of a directory tree. For more information, see “Getting Schema Information” on page 217 of Chapter 10, “Retrieving Server Information.”

An Introduction to LDAP

The Sun™ Java System Directory Server Resource Kit contains the LDAP SDK for C enabling programmers to write new, or modify existing, applications to connect to, search or update databases managed by directory servers using the Lightweight Directory Access Protocol (LDAP). This appendix offers a brief introduction to the LDAP and its concepts. It contains the following sections:

- Overview
- How Directory Services Work
- Defining Information Using LDAP
- Organizing LDAP Within a Directory Server
- Distinguished Names
- Data Distribution
- Additional Information

Overview

The LDAP is the Internet directory protocol. Developed at the University of Michigan at Ann Arbor in conjunction with the Internet Engineering Task Force (IETF), it is used for managing, and accessing data from, directory services. LDAP is based on a client/server model in which requests from a client program are transmitted over a network to a server program and results are transmitted back from the server to the client. LDAP servers provide the directory service, and LDAP clients use the directory service to access entries and attributes. It runs directly over the Transmission Control Protocol/Internet Protocol (TCP/IP).

NOTE

With LDAPv3, DN's should not have spaces after the comma. The format is accepted to be backward compatible with LDAPv2. As an example, the DN of Bab Jensen should be `uid=bjensen,ou=People,dc=example,dc=com` and not `uid=bjensen, ou=People, dc=example, dc=com`.

How Directory Services Work

A directory service is a distributed database application designed to manage the entries and attributes in a directory. It also makes the entries and attributes available to users and other applications. For example, a user might use the directory service to look up someone's telephone number. Another application might use the directory service to retrieve a list of email addresses. The Sun Java System Directory Server is an example of a directory service. It manages and provides information about users and organizational structures, such as groups and departments.

Defining Information Using LDAP

A directory database consists of data entries that contain descriptive information. These entries might describe people or network resources (such as printers or fax machines). The LDAP defines the types of data and units of information that can be stored in a directory. The directory is organized based on entries and attributes. An *entry*, the base unit of the LDAP, is a collection of attributes which contain information that describes it. Each *attribute* is comprised of a type and one or more values that describe a particular trait of the object's entry. The *type* describes what the information is; the *value* is the information itself. For example, attributes describing a person's entry might include one for the person's name (usually, common name, or `cn`), a telephone number, and an email address. Code Example A-1 is a sample entry taken from `testdata.ldif` in the `DSRK_base/java/LDAPSubtDel` directory. The text before the colon is the type; after the colon is the value. (For information on the LDAP Data Interchange Format, see "The LDIF File Format.")

Code Example A-1 Sample LDAP Entry

```
dn: uid=person5,ou=People,dc=siroe,dc=com
cn: person5
sn: person5
givenname: person5
objectclass: top
```


Code Example A-1 Sample LDAP Entry

```

objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
ou: Product Development
ou: People
l: Menlo Park
uid: person5
mail: person5@siroe.com
telephonenumber: 053-838-9384
facsimiletelephonenumber: 598-054-3814
roomnumber: 76
userpassword: secret

```

An attribute can have more than one value. Code Example A-2 shows how a person can have two common names (a formal name and a nickname) as well as two telephone numbers.

Code Example A-2 Entry for Jennifer Jensen

```

...
cn: Jennifer Jensen
cn: Jenny Jensen
mail: jen@corporation.com
telephoneNumber: 555-1213
telephoneNumber: 555-2059
roomNumber: 3996

```

The LDAP Schema

A base unit entry has a set of attribute types that are required and a set that can be used optionally. For example, the entry for a person is required to have a common name (cn) and a surname (sn). It is *allowed* to have a telephone number and an email address. This information is stored in a LDAP schema. Your directory server comes with a LDAP schema that is the definition of what can be stored in the directory. It includes *object classes* and *attributes*. The object classes are required or allowed and are defined with one or more attributes which are also required or allowed. Each attribute is associated with a syntax that determines what can be stored in that attribute (plain text, encoded data, etc.).

NOTE Attributes can also contain binary data. For example, attributes of a person might include a JPEG photo of the person or their voice recorded in an audio file format.

While the object classes and attributes defined in the schema should meet most of your needs, it can be necessary to extend the standard schema with additional object classes and / or attributes.

The LDIF File Format

LDAP Data Interchange Format (LDIF) is used to describe the contents and entries of a directory server in ASCII text format. LDIF is commonly used to build a directory database or to add large numbers of entries to the directory all at once. In addition, LDIF can also be used to describe changes to directory entries. For this reason, most of the directory server's command-line utilities rely on LDIF for input or output.

Each LDIF entry consists of a required distinguished name (dn), one or more object classes, and multiple attribute definitions. Code Example A-3 is the basic form of a directory entry represented in LDIF.

Code Example A-3 Generic LDIF Format

```
dn: <distinguished name>
objectClass: <object class>
objectClass: <object class>
...
<attribute type>[;lang-TAG]:<attribute value>
<attribute type>[;lang-TAG]:<attribute value>
...
```

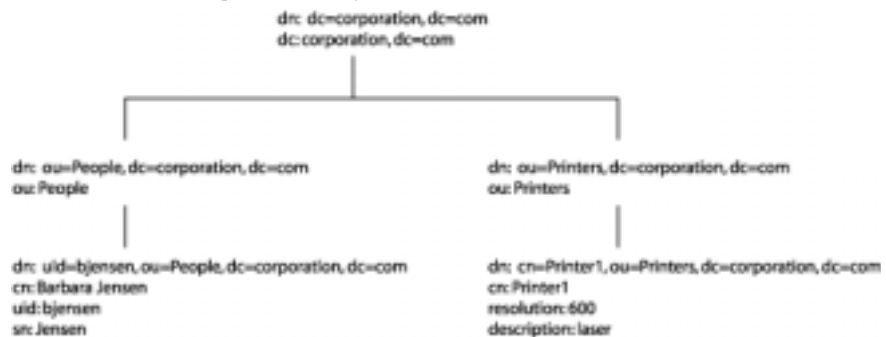
You can specify object classes and attributes in any order although the standard is to list all object classes first followed by the attribute type / value pairs. The space after the colon is optional.

Organizing LDAP Within a Directory Server

Because LDAP is intended for a global directory service, data is organized hierarchically, starting at a root and branching down into individual entries. At the top level of the hierarchy, entries represent large organizational units. Under these larger units might be entries for smaller organizational units. The hierarchy would siphon down to entries for individual people or resources.

Organizing and referring to your data within a directory server entails creating *containers* into which the base unit entries (explained in Defining Information Using LDAP) can be placed. For example, one container might hold all person entries while another might hold only those person entries defined as Human Resources personnel. Figure A-1 is a simple example of how entries might be organized within containers.

Figure A-1 Sample Hierarchy of Entries

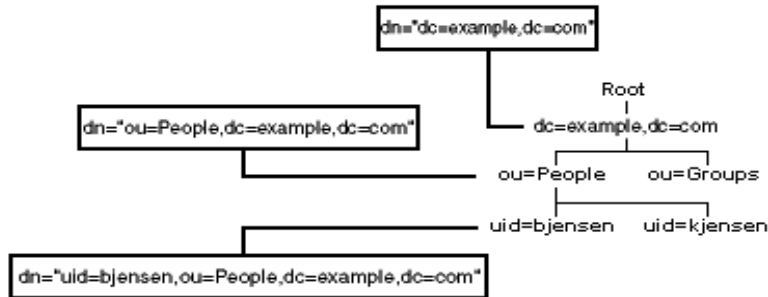


Distinguished Names

Each entry in a directory service is uniquely identified by a distinguished name (dn). The dn is a unique LDAP name that allows you to conclusively refer to any entry. For example, bjensen and kjensen are examples of user IDs that identify two entries at the same level. The dn includes a path of names that trace the entry back to the root of the tree. For the bjensen entry, the DN would be:

uid=bjensen,ou=People,dc=example,dc=com

Here, uid represents the user ID of the entry, ou represents the organizational unit to which the entry belongs, and dc represents the larger organization to which the entry belongs. Figure A-2 shows how distinguished names are used to identify entries uniquely in the directory hierarchy.

Figure A-2 DN to Identify Entries

Data Distribution

The data stored in a directory can be distributed among several LDAP servers. For example, one LDAP server for a corporation might contain entries representing North American organizational units and employees, while another might contain entries representing European organizational units and employees. Thus, if the North American LDAP server receives a request for information about an employee in the European branch, it can refer the request to the LDAP server in the European branch. In this way, LDAP servers appear to be a single source of directory information.

Additional Information

Additional information on the LDAP can be found in the:

- *Lightweight Directory Access Protocol (v3), RFC 2251*
(<http://www.faqs.org/rfcs/rfc2251.html>)
- *Sun ONE Directory Server Deployment Guide*
(<http://docs.sun.com/doc/816-6700-10>)

Additional information on the LDIF can be found in the:

- *The LDAP Data Interchange Format (LDIF) Technical Specification*
(<http://www.faqs.org/rfcs/rfc2849.html>)
- *Sun ONE Directory Server Reference Manual Appendix on LDIF*
(http://docs.sun.com/source/816-6699-10/ax_ldif.html)

A

- adding LDAP entries 149
- anonymous authentication 72
- applications
 - compiling 46
- asynchronous function
 - sample code 82
- asynchronous functions
 - add operation 156
 - and search results 118
 - calling 77
 - comparison operation 192
 - delete operation 176
 - examples 50
 - extended operation 268
 - modify operation 169
 - renaming operation 182
- attribute values
 - retrieving 128
- attributes
 - retrieving 127
- authentication
 - anonymous authentication 72
 - certificate-based 223
 - LDAP server 66
 - methods of 67
 - simple authentication 67
 - asynchronous 69
 - synchronous 68

B

- ber_alloc() 341
- ber_alloc_t() 341
- ber_bvdup() 343
- ber_bvecfree() 342
- ber_bvfree() 344
- ber_dup() 344
- ber_first_element() 345
- ber_flatten() 346
- ber_free() 346
- ber_get_boolean() 348
- ber_get_int() 348
- ber_get_next() 349
- ber_get_next_buffer() 350
- ber_get_next_buffer_ext() 350
- ber_get_null() 351
- ber_get_option() 351
- ber_get_stringa() 351
- ber_get_stringal() 352
- ber_get_stringb() 352
- ber_init() 352
- ber_init_w_nullchar() 353
- ber_next_element() 353
- ber_peek_tag() 354
- ber_printf() 354
- ber_put_bitstring() 356
- ber_put_boolean() 356
- ber_put_enum() 356
- ber_put_int() 356

- ber_put_null() 357
- ber_put_ostring() 357
- ber_put_seq() 357
- ber_put_set() 358
- ber_put_string() 358
- ber_read() 358
- ber_reset() 359
- ber_scanf() 359
- ber_set_option() 361
- ber_set_string_translators() 361
- ber_skip_tag() 361
- ber_sockbuf_alloc() 362
- ber_sockbuf_free() 362
- ber_sockbuf_free_data() 362
- ber_sockbuf_get_option() 362
- ber_sockbuf_set_option() 363
- ber_special_alloc() 363
- ber_special_free() 363
- ber_start_seq() 364
- ber_start_set() 364
- ber_svecfree() 364
- ber_write() 364
- BerElement 281
- berval 280
- bind
 - LDAP server 66
- binding
 - and referrals 96

C

- cache
 - in-memory 98
- calling asynchronous functions 77
- calling synchronous functions 76
- certificate-based authentication 223
- change DN of entry 181
- client-side sorting 131
 - by attribute 132
 - by attribute value 133
 - by multiple attributes 132

- closing LDAP server connection 73
- comparing LDAP entries 189
- compiling 46
- compiling applications 46
- controls 231
 - entry change notification 244
 - manage DSA IT 245
 - password policy 245
 - persistent search 242
 - proxied authorization control 246
 - server-side sorting 237
 - virtual list view 245

D

- data types 273
 - alphabetically 280
 - BerElement 281
 - berval 280
 - deprecated 274
 - FriendlyMap 281
 - LDAP 282
 - ldap_cache_fns 284
 - LDAP_CALLOC_CALLBACK 285
 - LDAP_CANCELPROC_CALLBACK 285
 - LDAP_CF_ADD_CALLBACK 285
 - LDAP_CF_BIND_CALLBACK 286
 - LDAP_CF_COMPARE_CALLBACK 286
 - LDAP_CF_DELETE_CALLBACK 286
 - LDAP_CF_FLUSH_CALLBACK 286
 - LDAP_CF_MODIFY_CALLBACK 286
 - LDAP_CF_MODRDN_CALLBACK 286
 - LDAP_CF_RESULT_CALLBACK 286
 - LDAP_CF_SEARCH_CALLBACK 286
 - LDAP_CF_UNBIND_CALLBACK 286
 - LDAP_CMP_CALLBACK 287
 - ldap_dns_fns 290
 - LDAP_DNSFN_GETHOSTBYADDR 288
 - LDAP_DNSFN_GETHOSTBYNAME 289
 - ldap_extra_thread_fns 291
 - LDAP_FREE_CALLBACK 295
 - ldap_io_fns 299
 - LDAP_IOF_CLOSE_CALLBACK 296
 - LDAP_IOF_CONNECT_CALLBACK 296

LDAP_IOF_IOCTL_CALLBACK 297
 LDAP_IOF_READ_CALLBACK 297
 LDAP_IOF_SELECT_CALLBACK 298
 LDAP_IOF_SOCKET_CALLBACK 298
 LDAP_IOF_SSL_ENABLE_CALLBACK 298
 LDAP_IOF_WRITE_CALLBACK 299
 LDAP_KEYCMP_CALLBACK 300
 LDAP_KEYFREE_CALLBACK 301
 LDAP_KEYGEN_CALLBACK 301
 LDAP_MALLOC_CALLBACK 301
 ldap_memalloc_fns 302
 LDAP_REALLOC_CALLBACK 306
 LDAP_REBINDPROC_CALLBACK 306
 LDAP_TF_GET_ERRNO_CALLBACK 308
 LDAP_TF_GET_LDERRNO_CALLBACK 309
 LDAP_TF_MUTEX_ALLOC_CALLBACK 310
 LDAP_TF_MUTEX_FREE_CALLBACK 311
 LDAP_TF_MUTEX_LOCK_CALLBACK 311
 LDAP_TF_MUTEX_TRYLOCK_CALLBACK 312
 LDAP_TF_MUTEX_UNLOCK_CALLBACK 312
 LDAP_TF_SEMA_ALLOC_CALLBACK 312
 LDAP_TF_SEMA_FREE_CALLBACK 313
 LDAP_TF_SEMA_POST_CALLBACK 313
 LDAP_TF_SEMA_WAIT_CALLBACK 313
 LDAP_TF_SET_ERRNO_CALLBACK 309
 LDAP_TF_SET_LDERRNO_CALLBACK 310
 LDAP_TF_THREADID_CALLBACK 313
 ldap_thread_fns 313
 LDAP_VALCMP_CALLBACK 317
 ldap_x_ext_io_fns 323
 LDAP_X_EXTIOF_CLOSE_CALLBACK 321
 LDAP_X_EXTIOF_CONNECT_CALLBACK 322
 LDAP_X_EXTIOF_DISPOSEHANDLE_CALLBA
 CK 322
 LDAP_X_EXTIOF_NEWHANDLE_CALLBACK
 323
 LDAP_X_EXTIOF_POLL_CALLBACK 324
 ldap_x_hostlist_status 324
 LDAP_X_PollFD 325
 LDAPAPIFeatureInfo 282
 LDAPAPIInfo 283
 LDAPControl 287
 LDAPFiltDesc 292
 LDAPFiltInfo 293
 LDAPHostEnt 295
 LDAPMemCache 302
 LDAPMessage 303

LDAPMod 304
 LDAPsortkey 307
 LDAPURLDesc 315
 LDAPVersion 318
 LDAPVirtualList 319
 summary by task 274
 typographical conventions 273
 deleting LDAP entry 174
 designing LDAP client 59
 determining LDAPv3 support 215
 directories 41
 Directory Server entry, see DSE
 DN
 retrieving 125
 documentation
 feedback 36
 terminology 33
 typographical conventions 33
 DSEs 211–215

E

entry change notification control 244
 error codes 635, 639–658
 setting 87
 errors
 reporting 87
 retrieving information 88
 SSL and 223
 example programs 49
 examples 49
 LDAP search 134
 extended operations 267–269

F

failover
 handling 100
 filter configuration file
 retrieving 143

filter configuration files 139
 free LDAPMessage structure 82
 FriendlyMap 281
 functions 327

- alphabetically 341
- ber_alloc() 341
- ber_alloc_t() 341
- ber_bvdup() 343
- ber_bvecfree() 342
- ber_bvfree() 344
- ber_dup() 344
- ber_first_element() 345
- ber_flatten() 346
- ber_free() 346
- ber_get_boolean() 348
- ber_get_int() 348
- ber_get_next() 349
- ber_get_next_buffer() 350
- ber_get_next_buffer_ext() 350
- ber_get_null() 351
- ber_get_option() 351
- ber_get_stringa() 351
- ber_get_stringal() 352
- ber_get_stringb() 352
- ber_init() 352
- ber_init_w_nullchar() 353
- ber_next_element() 353
- ber_peek_tag() 354
- ber_printf() 354
- ber_put_bitstring() 356
- ber_put_boolean() 356
- ber_put_enum() 356
- ber_put_int() 356
- ber_put_null() 357
- ber_put_ostring() 357
- ber_put_seq() 357
- ber_put_set() 358
- ber_put_string() 358
- ber_read() 358
- ber_reset() 359
- ber_scanf() 359
- ber_set_option() 361
- ber_set_string_translators() 361
- ber_skip_tag() 361
- ber_sockbuf_alloc() 362
- ber_sockbuf_free() 362
- ber_sockbuf_free_data() 362
- ber_sockbuf_get_option() 362
- ber_sockbuf_set_option() 363
- ber_special_alloc() 363
- ber_special_free() 363
- ber_start_seq() 364
- ber_start_set() 364
- ber_svecfree() 364
- ber_write() 364
- ldap_abandon() 365
- ldap_abandon_ext() 366
- ldap_add() 368
- ldap_add_ext() 371
- ldap_add_ext_s() 373
- ldap_add_s() 375
- ldap_ber_free() 377
- ldap_bind() 378
- ldap_bind_s() 379
- ldap_build_filter() 380
- ldap_cache_flush() 380
- ldap_charray_*() 381–382
- ldap_comapre_ext_s() 385
- ldap_compare() 382
- ldap_compare_ext() 383
- ldap_compare_s() 387
- ldap_control_free() 389
- ldap_controls_free() 389
- ldap_count_entries() 390
- ldap_count_messages() 391
- ldap_count_references() 392
- ldap_count_values() 393
- ldap_count_values_len() 395
- ldap_create_filter() 396
- ldap_create_geteffectiveRights_control() 400
- ldap_create_persistentsearch_control() 398
- ldap_create_proxiedauth_control() 401
- ldap_create_proxyauth_control() 402
- ldap_create_sort_control() 404
- ldap_create_sort_keylist() 405
- ldap_create_virtuallist_control() 407
- ldap_delete() 408
- ldap_delete_ext() 410
- ldap_delete_ext_s() 412
- ldap_delete_s() 414
- ldap_dn2ufn() 415
- ldap_entry2html() 416
- ldap_entry2html_search() 418
- ldap_entry2text() 419

ldap_entry2text_search() 421
 ldap_err2string() 423
 ldap_explode_dns() 425
 ldap_explode_rdn() 426
 ldap_explodedn() 424
 ldap_extended_operation() 427
 ldap_extended_operation_s() 429
 ldap_first_attribute() 431
 ldap_first_disptmpl() 433
 ldap_first_entry() 434
 ldap_first_message() 435
 ldap_first_reference() 436
 ldap_first_searchobj() 438
 ldap_first_tmplcol() 438
 ldap_first_tmplrow() 439
 ldap_free_friendlymap() 440
 ldap_free_searchprefs() 441
 ldap_free_sort_keylist() 441
 ldap_free_templates() 442
 ldap_free_urldesc() 443
 ldap_friendly_name() 444
 ldap_get_dn() 446
 ldap_get_entry_controls() 448
 ldap_get_lang_values() 452
 ldap_get_lang_values_len() 453
 ldap_get_lderrno() 455
 ldap_get_option() 457
 ldap_get_values() 463
 ldap_get_values_len() 465
 ldap_getfilter_free() 449
 ldap_getfirstfilter() 450
 ldap_getnextfilter() 456
 ldap_init() 467
 ldap_init_getfilter() 470
 ldap_init_getfilter_buf() 471
 ldap_init_searchprefs() 472
 ldap_init_searchprefs_buf() 473
 ldap_init_templates() 474
 ldap_init_templates_buf() 475
 ldap_is_dns_dn() 476
 ldap_is_ldap_url() 477
 ldap_keysort_entries() 478
 ldap_ldapssl_*() 579
 ldap_memcache_destroy() 479
 ldap_memcache_flush() 480
 ldap_memcache_get() 480
 ldap_memcache_init() 481
 ldap_memcache_set() 483
 ldap_memcache_update() 484
 ldap_memfree() 485
 ldap_modify() 486
 ldap_modify_ext() 488
 ldap_modify_ext_s() 490
 ldap_modify_s() 493
 ldap_modrdn() 495
 ldap_modrdn_s() 495
 ldap_modrdn2() 495
 ldap_modrdn2_s() 497
 ldap_mods_free() 498
 ldap_msgfree() 500
 ldap_msgid() 502
 ldap_msgtype() 503
 ldap_multisort_entries() 505
 ldap_name2template() 507
 ldap_next_attribute() 507
 ldap_next_disptmpl() 509
 ldap_next_entry() 510
 ldap_next_message() 511
 ldap_next_reference() 512
 ldap_next_searchobj() 513
 ldap_next_tmplcol() 514
 ldap_next_tmplrow() 515
 ldap_oc2template() 515
 ldap_open() 516
 ldap_parse_entrychange_control() 517
 ldap_parse_extended_result() 519
 ldap_parse_reference() 521
 ldap_parse_result() 522
 ldap_parse_sasl_bind_result() 525
 ldap_parse_sort_control() 527
 ldap_parse_virtuallist_control() 528
 ldap_perror() 530
 ldap_rename() 531
 ldap_rename_s() 534
 ldap_result() 536
 ldap_result2error() 539
 ldap_sasl_bind() 540
 ldap_sasl_bind_s() 542
 ldap_search() 544
 ldap_search_ext() 548
 ldap_search_ext_s() 550
 ldap_search_s() 553
 ldap_search_st() 555
 ldap_set_filter_additions() 557

- ldap_set_lderrno() 559
- ldap_set_option() 560
- ldap_set_rebind_proc() 565
- ldap_setfilteraffixes() 558
- ldap_simple_bind() 569
- ldap_simple_bind_s() 572
- ldap_sort_entries() 574
- ldap_sort_strcasecmp() 576
- ldap_sort_values() 577
- ldap_str2chararray() 594
- ldap_tmplatrrs() 594
- ldap_tmplerr2string() 595
- ldap_ufn_*() 596
- ldap_unbind() 597
- ldap_unbind_ext() 600
- ldap_unbind_s() 598
- ldap_url_parse() 601
- ldap_url_parse_no_defaults() 603
- ldap_url_search() 604
- ldap_url_search_s() 608
- ldap_url_search_st() 609
- ldap_utf8characters() 611
- ldap_utf8copy() 612
- ldap_utf8getcc() 612
- ldap_utf8isalnum() 613
- ldap_utf8isalpha() 613
- ldap_utf8isdigit() 614
- ldap_utf8isspace() 614
- ldap_utf8len() 614
- ldap_utf8next() 615
- ldap_utf8prev() 615
- ldap_utf8strtok_r() 616
- ldap_vals2html() 616
- ldap_vals2text() 618
- ldap_value_free() 619
- ldap_value_free_len() 620
- ldap_version() 621
- ldap_x_*() 621
- prldap_*() 625
- summary by task 328

G

- getting 217
- getting schema information 217

H

- handling failover 100
- handling referrals 93
- header files 42

I

- I/O functions
 - and SSL 223
- information sources
 - related to DSRK 34
 - related to Sun products 34
 - related to third-party products 35
 - Mozilla Project 35
 - Perl development 35
- initializing LDAP session 61
- initializing LDAP session sample code 63
- installed directories 41

L

- LDAP Application Programming Interface (API) 40
 - asynchronous functions 40
 - synchronous functions 40
- LDAP client
 - designing 59
 - sample 53–56
- LDAP controls, *see* controls
- LDAP Data Interchange Format, *see* LDIF
- LDAP data type 282
- LDAP directory
 - searching 105
- LDAP entries
 - adding 149
 - comparing 189
 - modifying 161
- LDAP entry
 - change DN 181
 - delete 174
 - retrieving attribute values 128

- retrieving attributes 127
- retrieving DN 125
- LDAP operations 73
- LDAP search
 - examples 134
- LDAP server
 - and SSL 221
 - anonymous authentication 72
 - binding and authenticating 66
 - closing connection 73
 - LDAP operations 73
 - methods of authentication 67
 - simple authentication 67
 - asynchronous 69
 - synchronous 68
- LDAP session
 - initializing 61
 - sample code 63
 - setting preferences 64
- LDAP URLs 199–207
- ldap.h header file
 - including 43
- ldap_abandon() 365
- ldap_abandon_ext() 366
- ldap_add() 368
- ldap_add_ext() 371
- ldap_add_ext_s() 373
- ldap_add_s() 375
- ldap_ber_free() 377
- ldap_bind() 378
- ldap_bind_s() 379
- ldap_build_filter() 380
- ldap_cache_flush() 380
- ldap_cache_fns 284
- LDAP_CALL 97
- LDAP_CALLBACK 97
- LDAP_CALLOC_CALLBACK 285
- LDAP_CANCELPROC_CALLBACK 285
- LDAP_CF_ADD_CALLBACK 285
- LDAP_CF_BIND_CALLBACK 286
- LDAP_CF_COMPARE_CALLBACK 286
- LDAP_CF_DELETE_CALLBACK 286
- LDAP_CF_FLUSH_CALLBACK 286
- LDAP_CF_MODIFY_CALLBACK 286
- LDAP_CF_MODRDN_CALLBACK 286
- LDAP_CF_RESULT_CALLBACK 286
- LDAP_CF_SEARCH_CALLBACK 286
- LDAP_CF_UNBIND_CALLBACK 286
- ldap_charray_*() 381–382
- LDAP_CMP_CALLBACK 287
- ldap_comapre_s() 387
- ldap_compare() 382
- ldap_compare_ext() 383
- ldap_compare_ext_s() 385
- ldap_control_free() 389
- ldap_controls_free() 389
- ldap_count_entries() 390
- ldap_count_messages() 391
- ldap_count_references() 392
- ldap_count_values() 393
- ldap_count_values_len() 395
- ldap_create_filter() 396
- ldap_create_geteffectiveRights_control() 400
- ldap_create_persistentsearch_control() 398
- ldap_create_proxiedauth_control() 401
- ldap_create_proxyauth_control() 402
- ldap_create_sort_control() 404
- ldap_create_sort_keylist() 405
- ldap_create_virtuallist_control() 407
- ldap_delete() 408
- ldap_delete_ext() 410
- ldap_delete_ext_s() 412
- ldap_delete_s() 414
- ldap_dn2ufn() 415
- ldap_dns_fns 290
- LDAP_DNSFN_GETHOSTBYADDR 288
- LDAP_DNSFN_GETHOSTBYNAME 289
- ldap_entry2html() 416
- ldap_entry2html_search() 418
- ldap_entry2text() 419
- ldap_entry2text_search() 421
- ldap_err2string() 423
- ldap_explode_dns() 425
- ldap_explode_rdn() 426
- ldap_explodedn() 424

ldap_extended_operation()	427	LDAP_IOF_SOCKET_CALLBACK	298
ldap_extended_operation_s()	429	LDAP_IOF_SSL_ENABLE_CALLBACK	298
ldap_extra_thread_fns	291	LDAP_IOF_WRITE_CALLBACK	299
ldap_first_attribute()	431	ldap_is_dns_dn()	476
ldap_first_disptmpl()	433	ldap_is_ldap_url()	477
ldap_first_entry()	434	LDAP_KEYCMP__CALLBACK	300
ldap_first_message()	435	LDAP_KEYFREE__CALLBACK	301
ldap_first_reference()	436	LDAP_KEYGEN__CALLBACK	301
ldap_first_searchobj()	438	ldap_keysort_entries()	478
ldap_first_tmplcol()	438	ldap_ldapssl_*()	579
ldap_first_tmplrow()	439	LDAP_MALLOC__CALLBACK	301
LDAP_FREE_CALLBACK	295	ldap_memalloc_fns	302
ldap_free_friendlymap()	440	ldap_memcache_destroy()	479
ldap_free_searchprefs()	441	ldap_memcache_flush()	480
ldap_free_sort_keylist()	441	ldap_memcache_get()	480
ldap_free_templates()	442	ldap_memcache_init()	481
ldap_free_urldesc()	443	ldap_memcache_set()	483
ldap_friendly_name()	444	ldap_memcache_update()	484
ldap_get_dn()	446	ldap_memfree()	485
ldap_get_entry_controls()	448	ldap_modify()	486
ldap_get_lang_values()	452	ldap_modify_ext()	488
ldap_get_lang_values_len()	453	ldap_modify_ext_s()	490
ldap_get_lderrno()	455	ldap_modify_s()	493
ldap_get_option()	457	ldap_modrdn()	495
ldap_get_values()	463	ldap_modrdn_s()	495
ldap_get_values_len()	465	ldap_modrdn2()	495
ldap_getfilter_free()	449	ldap_modrdn2_s()	497
ldap_getfirstfilter()	450	ldap_mods_free()	498
ldap_getnextfilter()	456	ldap_msgfree()	500
ldap_init()	467	ldap_msgid()	502
ldap_init_getfilter()	470	ldap_msgtype()	503
ldap_init_getfilter_buf()	471	ldap_multisort_entries()	505
ldap_init_searchprefs()	472	ldap_name2template()	507
ldap_init_searchprefs_buf()	473	ldap_next_attribute()	507
ldap_init_templates()	474	ldap_next_disptmpl()	509
ldap_init_templates_buf()	475	ldap_next_entry()	510
ldap_io_fns	299	ldap_next_message()	511
LDAP_IOF_CLOSE_CALLBACK	296	ldap_next_reference()	512
LDAP_IOF_CONNECT_CALLBACK	296	ldap_next_searchobj()	513
LDAP_IOF_IOCTL_CALLBACK	297	ldap_next_tmplcol()	514
LDAP_IOF_READ_CALLBACK	297	ldap_next_tmplrow()	515
LDAP_IOF_SELECT_CALLBACK	298	ldap_oc2template()	515

ldap_open()	516	LDAP_TF_SEMA_ALLOC_CALLBACK	312
ldap_parse_entrychange_control()	517	LDAP_TF_SEMA_FREE_CALLBACK	313
ldap_parse_extended_result()	519	LDAP_TF_SEMA_POST_CALLBACK	313
ldap_parse_reference()	521	LDAP_TF_SEMA_WAIT_CALLBACK	313
ldap_parse_result()	522	LDAP_TF_SET_ERRNO_CALLBACK	309
ldap_parse_sasl_bind_result()	525	LDAP_TF_SET_LDERRNO_CALLBACK	310
ldap_parse_sort_control()	527	LDAP_TF_THREADID_CALLBACK	313
ldap_parse_virtuallist_control()	528	ldap_thread_fns	313
ldap_perror()	530	ldap_tmplattrs()	594
LDAP_REALLOC_CALLBACK	306	ldap_tmplerr2string()	595
LDAP_REBINDPROC_CALLBACK	306	ldap_ufn_*	596
ldap_rename()	531	ldap_unbind()	597
ldap_rename_s()	534	ldap_unbind_ext()	600
ldap_result()	536	ldap_unbind_s()	598
ldap_result2error()	539	ldap_url_parse()	601
ldap_sasl_bind()	540	ldap_url_parse_no_defaults()	603
ldap_sasl_bind_s()	542	ldap_url_search()	604
ldap_search()	544	ldap_url_search_s()	608
ldap_search_ext()	548	ldap_url_search_st()	609
ldap_search_ext_s()	550	ldap_utf8characters()	611
ldap_search_s()	553	ldap_utf8copy()	612
ldap_search_st()	555	ldap_utf8getccc()	612
ldap_set_filter_additions()	557	ldap_utf8isalnum()	613
ldap_set_lderrno()	559	ldap_utf8isalpha()	613
ldap_set_option()	560	ldap_utf8isdigit()	614
ldap_set_rebind_proc()	565	ldap_utf8isspace()	614
ldap_setfilteraffixes()	558	ldap_utf8len()	614
ldap_simple_bind()	569	ldap_utf8next()	615
ldap_simple_bind_s()	572	ldap_utf8prev()	615
ldap_sort_entries()	574	ldap_utf8strtok_r()	616
ldap_sort_strcasecmp()	576	LDAP_VALCMP_CALLBACK	317
ldap_sort_values()	577	ldap_vals2html()	616
ldap_ssl.h header file		ldap_vals2text()	618
including	43	ldap_value_free()	619
ldap_str2charray()	594	ldap_value_free_len()	620
LDAP_TF_GET_ERRNO_CALLBACK	308	ldap_version()	621
LDAP_TF_GET_LDERRNO_CALLBACK	309	ldap_x_*	621
LDAP_TF_MUTEX_ALLOC_CALLBACK	310	ldap_x_ext_io_fns	323
LDAP_TF_MUTEX_FREE_CALLBACK	311	LDAP_X_EXTIOF_CLOSE_CALLBACK	321
LDAP_TF_MUTEX_LOCK_CALLBACK	311	LDAP_X_EXTIOF_CONNECT_CALLBACK	322
LDAP_TF_MUTEX_TRYLOCK_CALLBACK	312	LDAP_X_EXTIOF_DISPOSEHANDLE_CALLBACK	322
LDAP_TF_MUTEX_UNLOCK_CALLBACK	312		

- LDAP_X_EXTIOF_NEWHANDLE_CALLBACK 323
- LDAP_X_EXTIOF_POLL_CALLBACK 324
- ldap_x_hostlist_status 324
- LDAP_X_PollFD 325
- LDAPAPIFeatureInfo 282
- LDAPAPIInfo 283
- LDAPControl 287
- LDAPFiltDesc 292
- LDAPFiltInfo 293
- LDAPHostEnt 295
- LDAPMemCache 302
- LDAPMessage 303
 - free structure 82
- LDAPMod 304
- LDAPMod structures 147
- LDAPSortkey 307
- LDAPURLDesc 315
- LDAPv3 support
 - determining 215
- LDAPVersion 318
- LDAPVirtualList 319
- libraries 44
- Lightweight Directory Access Protocol, see LDAP

M

- manage DSA IT control 245
- managing memory 86
- memory
 - freeing search filters from 145
 - managing 86
- modifying LDAP entries 161
- Mozilla Project 35
- multithreaded clients 251–265

N

- nsldap32v30.dll file 48

- nsldapssl32v30.lib file
 - linking to 47

O

- operational attributes 112

P

- password policy control 245
- patches
 - Solaris 35
- Perl development 35
- persistent search control 242
- preferences
 - restart session 65
 - search 112
 - setting session 64
- prldap_*() 625
- proxied authorization control 246
- pthreads client 255

R

- referral hop limit 95
- referrals
 - and binding 96
 - and search references 94
 - from search references 130
 - handling 93
 - referral hop limit 95
- reporting errors 87
- restart session preference 65
- result codes 635, 639–658
- retrieve attributes in search 111
- retrieving attribute values 128
- retrieving attributes 127
- retrieving DN 125

retrieving error information 88
 retrieving SDK information 85

S

sample code
 initializing LDAP session 63
 sample LDAP client 53–56
 SASL 225–229
 schema information 217
 SDK information 85
 search filters 109
 creating programmatically 146
 freeing from memory 145
 parameters 142
 prefixes 145
 suffixes 145
 search LDAP directory 105
 client-side sorting 131
 parameters 107
 preferences 112
 request 106
 results 114
 retrieve attributes 111
 search filters 109
 server-side sorting 131
 sorting results 131
 search preferences 112
 search references 94
 retrieving from referrals 130
 search request 106
 filters 109
 parameters 107
 search results 114
 and asynchronous functions 118
 and synchronous functions 115
 client-side sorting 131
 server-side sorting 131
 sorting 131
 Secure Sockets Layer, see SSL
 semaphore 253
 server-side sorting 131
 server-side sorting control 237

setting error codes 87
 setting LDAP session preferences 64
 setting up cache 98
 simple authentication 67
 asynchronous 69
 synchronous 68
 Simple Authentication and Security Layer, see SASL
 Solaris
 patches 35
 support 35
 SSL
 and I/O functions 223
 and the SDK 219
 connect to LDAP server 221
 handling errors 223
 support
 Solaris 35
 synchronous functions
 add operation 153
 and search results 115
 calling 76
 comparison operation 190
 delete operation 174
 examples 49
 extended operation 268
 renaming operation 181

T

thread functions 251

V

virtual list view control 245

